Machine Learning I

Project: *Android Malware Detection*

Master in Artificial Intelligence 2023/2024

Group C

Abel Juncal Suárez
Alejandro Dopico Castro
Javier Torón Artiles
Lucía María Álvarez Crespo

December 14, 2023

Abstract

The digital world is highly interconnected, and the constant exchange of information between different devices and applications has led to an increased risk of security threats, especially on Android devices. The detection and prevention of Android malware has become critical because it can compromise user privacy, device security, and essential functions like communication and digital transactions. This study examines and compares various techniques to detect malware on Android devices. Multiple machine learning-based methods were explored due to the growing risks that malware poses to the security and privacy of Android users. We conducted comprehensive experiments using representative metrics to evaluate the strengths and limitations of each technique. The results showed the relative effectiveness of each method in detecting various malware variants, highlighting the need for comprehensive and adaptive approaches to combat the constant evolution of malware threats on Android devices.

KEYWORDS: *Android, Malware, Detection, KNN, ANN, SVM, Decision Tree, Ensembles, Machine Learning*

1 Introduction

Android devices have a dominant market share but are also vulnerable to cyber threats. Traditional manual methods for malware analysis are time-consuming and cannot scale effectively. Machine learning-based automated analysis techniques have shown promise in detecting Android malware. This work aims to enhance these techniques and contribute to the development of efficient, scalable, and adaptive methods to safeguard Android users against evolving malware threats.

1.1 Motivation

Android is currently the most widely used smartphone operating system, holding a market share of over 70% since October 2016, as per a Statista[1] report [2]. However, the popularity and openness of this operating system make it a prime target for cyberattacks [5]. Malicious developers can exploit vulnerabilities in the system to release harmful applications that can compromise user privacy or perform other dangerous actions on the users' mobile devices. This puts the safety of mobile device users at risk.

Android malware defence is a crucial research area in computer security. Analyzing malware manually, by creating corresponding rules and inspecting the behaviours and source code of suspicious Android apps, is a time-consuming process and cannot be scaled to a large amount of Android software. Moreover, with malware techniques constantly evolving [8], manual malware analysis is unable to keep up with the evolving attack strategies. In recent years, there has been a significant amount of research related to automatic Android malware analysis. These studies have utilized data mining and machine learning approaches to achieve acceptable malware detection performance [4]. These approaches utilize a series of machine learning algorithms, such as support vector machine, k-NN, artificial neural networks, and decision trees, to build a prediction model based on feature vectors extracted from a selected dataset.

The increasing use of Android OS has raised concerns about security. However, researchers are making progress in countering malware threats through automated analysis. With the help of machine learning techniques, researchers are developing scalable solutions to combat these threats. By using a variety of machine learning algorithms and feature-rich datasets, these automated approaches have shown promising results in detecting malware. As the threat landscape continues to evolve, it is essential to refine these methodologies to protect mobile device users from sophisticated cyber threats and maintain the security of the Android ecosystem.

[1] https://www.statista.com/

1.2 State of the art

This field is a really interesting area to investigate due to the importance and impact that it can have nowadays. Therefore, there exist many diverse studies related to this area of analysis, which investigate the approaches to the detection of malware using machine learning. These articles do not only focus on the same topics in relation to this area, but they cover different types of subjects.

The investigation sheds light on a significant challenge faced by Android Malware classifiers, known as model drift. The issue arises when classifiers trained on a year's worth of data become outdated due to the constant evolution of malware. Malicious actors continually introduce new malware types or functionalities to avoid detection by existing models [3]. Additionally, updates in benign apps can also confuse these models. The study aims to simplify the laborious labelling process for analysts dealing with emerging challenges in classifying both malicious and benign apps.

According to some investigators, it is not enough to determine if an app is malicious or benign through binary classification [9]. Instead, it is important to identify the specific type of malicious behavior it exhibits. The goal is not just to achieve high accuracy, but also to interpret the results. To achieve this, researchers are using a Multi-layer Perceptron (MLP) model with a customized attention mechanism to interpret the malicious behaviors of malware found in Android apps. Emphasis is placed on understanding and interpreting what the model has learned and how it makes predictions.

Certain studies have focused on a technique called DREBIN [1], which enables detection of a malicious Android application directly from the smartphone. The main objectives of this research were to develop a method that could identify Android malware by leveraging the power of machine learning and static analysis, to trace back the patterns of the indicative features, and to detect the malware from the smartphone itself.

Some studies refer to the development of SeqMobile [6], which makes use of behavior-based sequences and deep neural networks, in order to identify malware in Android smartphones. The main objectives of this project are to create a system that can accurately and efficiently detect malware on Android devices, extract semantic feature sequences from binary files to achieve high classification accuracy, and remove any repeated elements from the sequences while ensuring optimal performance.

The studies mentioned earlier represent the different approaches researchers have taken in the field of Android malware detection. These investigations focus on various aspects of the topic, using innovative methods and models to address the challenges faced by security analysts dealing with the complex landscape of Android malware. The insights gained from each study are unique and help in understanding the evolving threat landscape. It is crucial to have a comprehensive understanding and interpretation of the malicious behaviours exhibited in Android apps. The exploration of these behaviours goes beyond binary classification and aims to provide a nuanced comprehension of the intricate nuances of malware activities.

1.3 Dataset explanation

The chosen dataset[2] comprises 355,630 rows and 86 columns, forming a substantial dataset for multiclass classification. In this case, the classification task involves sorting instances into four distinct classes: *Android Adware*, *Android Scareware*, *Android SMS Malware*, and *Benign*.

Acquired from the CIC repository [7], this dataset features a comprehensive collection of entries representing various aspects related to Android applications. Each entry contains information across 85 columns, encompassing diverse attributes and features pertinent to the classification task.

The dataset represents network flow characteristics and statistics related to Android application behavior. It includes columns detailing attributes such as flow identifiers, IP addresses (source and destination), ports used, protocol information, packet details (length, count), flow duration, various statistical measures, and flags. Additionally, it contains columns like 'Label' specifying the classification into *Android Ad-*

[2] https://www.kaggle.com/datasets/subhajournal/android-malware-detection

ware, *Android Scareware*, *Android SMS Malware*, or *Benign* categories based on the flow attributes.

Given the immense size of the original dataset, a decision was made to derive a representative subset comprising 1% of the entire dataset. This subsampling was conducted using a stratified approach, ensuring the preservation of the same distribution proportions found in the complete dataset across its various classes. This strategic downsizing was implemented to expedite processing times and facilitate cross-validation training procedures. By working with this reduced yet representative subset, it allows for more efficient model training and validation without compromising the integrity of the dataset's class distribution.

Additionally, preliminary testing conducted on this 1% subset indicated that the results obtained mirrored those attained from the analysis of the complete dataset. This preliminary evaluation demonstrated that despite the reduction in dataset size, the outcomes and trends observed remained consistent with those derived from the entire dataset. This confirmation suggests that the smaller subset accurately captures the essential characteristics and patterns present in the larger dataset, affirming its suitability for expedited testing and validation processes without compromising the fidelity of the results.

Label	Entries
Android_Adware	1,475
Android_Scareware	1,171
Android_SMS_Malware	674
Benign	237
Total size	3,557

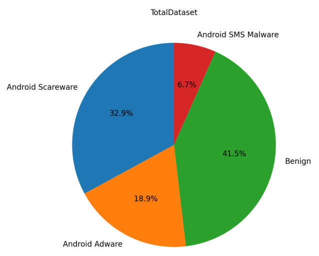

Figure 1: Label distribution of the dataset

The provided table (Table 1), presents a breakdown of entries within the dataset across various categories. It delineates the distribution of instances into the four distinct classes. Each category exhibits a different count of entries. This tabulation provides an insightful overview of the dataset's composition, illustrating the distribution among the different classes, which is vital for understanding the dataset's class balance and proportions.

It is apparent from the table that the dataset is heavily imbalanced. The benign class makes up only 7% of the entire dataset. This significant imbalance among classes, with considerably fewer instances in the benign category compared to the malware classes, poses a significant challenge for machine learning models. Such an imbalance may cause the models to show biases towards the majority classes in their predictions.

1.4 About the metrics to be used

The metrics chosen for evaluation include accuracy, recall (sensitivity), specificity, and the F1-score.

- **Accuracy:** Calculates the proportion of correctly predicted instances among the total number of instances in the dataset. It provides a general measure of the model's correctness across all classes.

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

- **Recall (Sensitivity):** Measures the ratio of correctly predicted positive observations to the total actual positives. It focuses on the ability of the model to correctly identify all positive instances.

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

- **Specificity:** Calculates the ratio of correctly predicted negative observations to the total actual negatives. It concentrates on the model's ability to correctly identify all negative instances.

$$\text{Specificity} = \frac{TN}{TN + FP}$$

- **F1 Score:** The harmonic mean of precision and recall. It balances precision (correctly identifying positive instances among all predicted positives) and recall (capturing all actual positives). Particularly valuable in imbalanced datasets.

$$\text{F1 Score} = 2 \times \frac{Accuracy \times Recall}{Accuracy + Recall}$$

In the initial binary classification approaches aimed at distinguishing between benign and malware instances (or malware vs non-malware) , recall and specificity are crucial. These metrics together provide a comprehensive understanding of the model's performance in such binary classifications.

In the context of an imbalanced dataset, the F1 score becomes particularly valuable as it considers both precision and recall, providing a balanced assessment of model performance even when classes are disproportionate in size.

All metrics are being calculated using a weighted approach, accounting for class imbalances by considering each class's contribution to the overall metric proportional to its representation in the dataset. This ensures a fair and representative evaluation of the model's performance across all classes.

1.5 Code structure

This section delineates the organizational structure and files constituting the codebase. The code is compartmentalized across multiple Julia files, amalgamating both freshly developed and pre-existing code from the course. Moreover, a pivotal Julia script, `main.jl` orchestrates the invocation of other modules, driving the entire workflow.

Central to this codebase is the `main.jl` script, functioning as the primary entry point. Additionally, a dedicated file script was generated for each approach, denoted by the approach name (i.e `first_approach.jl`. The codebase is augmented by a `utils` package housing four distinct modules:

- `ClassificationMetrics`: Tailored for evaluating classification model efficacy and precision, this module furnishes a suite of indispensable functions. These functions enable the classification of model outputs using specified thresholds, computation of accuracy metrics for both binary and multiclass scenarios, and comprehensive confusion matrix generation. This suite facilitates exhaustive analysis of model predictions, illuminating their predictive capabilities comprehensively.

- `DataHandling`: Designed to streamline data management, this module expedites data manipulation and partitioning for machine learning tasks. It offers utilities for partitioning datasets into training and test sets, streamlining dataset management for training, validation, and essential data-centric tasks. This module plays a pivotal role in organizing and preparing data, optimizing it for model training and evaluation phases.

- `DataPreprocessing`: Specializing in transforming raw data into a format conducive to machine learning models, this module encompasses tailored functions. These functions encode categorical features, compute normalization parameters (e.g., min-max, zero-mean normalization), and apply these transformations to datasets. This preparatory toolkit ensures data compatibility with machine learning algorithms by executing crucial data transformations.

- `Model`: Serving as a comprehensive suite for training machine learning models, this module encompasses diverse functionalities. It aids in constructing models, conducting training iterations with varied parameter configurations, creating model ensembles, performing cross-validation techniques for assessing generalizability, and managing the training process for individual models and ensembles. By furnishing these tools, this module facilitates iterative enhancement and optimization of machine learning models.

In our development workflow, we follow a strong coding convention and formatting style called `JuliaBlue`[3] To ensure consistency and adherence to the prescribed coding standards across the entire codebase, we have implemented a pre-commit[4] hook that automatically formats the code according to this style guide before any commits are made to the repository. This approach helps us maintain a consistent code structure throughout the development process.

2 Experimentation

This section provides a brief description of each approach's characteristics, execution results, and comments.

2.1 Approach 1: Binary Classification

In the initial binary classification approach, the objective is to discern the presence of malware against benign instances. This approach focuses on identifying whether an instance represents malicious behavior or benign characteristics within the dataset. By assigning a binary label to each instance—0 for benign and 1 for malware—the classification task aims to develop a model capable of accurately distinguishing between these two distinct classes. The emphasis lies in effectively detecting and differentiating malicious behavior from benign activity, forming the foundational step towards building robust and accurate malware detection systems.

The challenge of an imbalanced class, particularly with the benign instances representing only around 7% of the dataset, poses a significant issue in training robust and accurate models. This severe class imbalance, where the benign class is substantially underrepresented compared to the malware class, can lead to biased models that favor predicting the majority class, i.e., malware, while overlooking the nuances of the minority class, in this case, benign instances. Such imbalance might cause the model to be less sensitive to detecting benign behavior, potentially resulting in a higher rate of false negatives in benign predictions. Addressing this imbalance becomes crucial as it directly impacts the model's ability to effectively learn and generalize patterns from the dataset, emphasizing the need for specialized techniques to mitigate the effects of class imbalance during model training and evaluation.

Table 2 presents the label distribution within the dataset for the initial approach. It highlights the stark class imbalance, notably showing 237 entries labeled as Benign compared to 3,320 instances classified as Malware. This imbalance underscores the challenge posed by the significantly smaller representation of benign instances compared to malware, with the total dataset size amounting to 3,557 entries.

Label	Entries
Benign	237
Malware	3,320
Total size	3,557

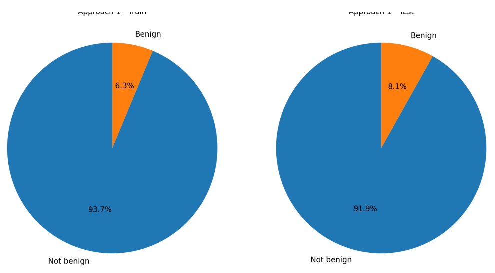

Figure 2: Label distribution (first approach)

[3] https://github.com/invenia/BlueStyle
[4] https://pre-commit.com/

2.1.1 Data Preprocessing

All input columns underwent normalization using a min-max technique and were subsequently converted to `float32` format. This preprocessing step standardized the data, scaling each feature to a common range.

The choice of employing min-max scaling stems from its ability to preserve the relationships and distributions within the data while constraining all features to a common range, typically between 0 and 1. This scaling method proves beneficial in maintaining the original data structure and relative differences between feature values, making it suitable for scenarios where preserving these relationships is crucial, especially in machine learning models sensitive to feature scales.

Moreover, extracting the minimum and maximum parameters from the training dataset and applying them to the test set ensures consistency and avoids data leakage. By using the training set's statistics for scaling, the model remains blind to the test set during preprocessing, preventing any information about the test data from influencing the transformation process. This approach promotes a more accurate estimation of the model's true generalization performance by ensuring that the scaling procedure is based solely on the training data's characteristics.

2.1.2 Other data related to the experiment

Cross Validation It was performed on a 10 fold cross-validation. Employing a *k* value of 10 proves advantageous, especially when dealing with a dataset of around 3,000 rows. This choice of *k* in the cross-validation process ensures a robust evaluation of the model's performance by partitioning the dataset into 10 equally sized subsets. With each iteration, 9 subsets are utilized for training the model, while the remaining subset is employed for validation, repeating this process 10 times to cover the entire dataset systematically.

This approach allows for a comprehensive assessment of the model's generalization capabilities, leveraging a significant portion of the data for both training and validation across multiple iterations. Utilizing a *k* value of 10 strikes a balance between computational efficiency and obtaining reliable estimates of the model's performance in scenarios with a moderately sized dataset.

Train-test split The test set was made using a hold-out strategy, segregating 20% of the dataset to serve as an independent test subset. This separation ensured that the model's performance could be objectively assessed on unseen data, distinct from the training set, validating its generalizability and efficacy in real-world scenarios.

Feature engineering Feature engineering and data cleaning were integral parts of the preprocessing stage. Initially, the Flow ID and Timestamp columns were eliminated due to their lack of contribution to the classification problem. Additionally, the high cardinality of the Flow ID column rendered it less informative for our task. Columns showcasing a singular value across the dataset, indicating no variability, were also removed. Notably, null values and duplicates, fortuitously, were absent in our dataset, thus necessitating no further cleaning actions regarding these issues.

Ultimately, the Source IP and Destination IP columns were transformed from text strings to decimal representations employing an octet structure. This conversion was done with the purpose of obtaining more readable and structured information, as it can sometimes facilitate the identification of possible attacks through the inspection of this specific traffic.

After the aforementioned preprocessing steps, the dataset consolidated to a final structure comprising 70 input columns. These columns encompassed the refined and engineered features, ready to be utilized for subsequent modeling and analysis.

ANN hyperparameters For the ANN training, a set of predefined hyperparameters was utilized to maintain consistency and facilitate model convergence. These fixed hyperparameters included a maximum of 200 epochs for training, a learning rate set at 0.01 to regulate the step size during gradient descent, and a validation set comprising 30 epochs for early stopping mechanisms. Additionally, 30 repetitions were performed to ensure robustness and capture variations in the training process, accompanied by a validation ratio of 0.1 to allocate a portion of the data for validation for Early Stopping. The transfer functions employed were filled with (sigmoid activation function) across the deep dense layers. However, for the final model iteration, the number of repetitions was curtailed to 1, mimicking a real-time scenario to expedite the training process while maintaining a semblance of a practical model implementation.

Model selection for ensembles criteria During the model selection phase for ensemble learning comprising both hard voting and stacking techniques, the criteria for inclusion revolved around maximizing specificity, particularly crucial for the minority class. The three models—K-Nearest Neighbors (KNN), Support Vector Machine (SVM), and Decision Tree (DT)—were meticulously tuned to yield the highest specificity scores. In scenarios where multiple models demonstrated equal specificity, preference was accorded to the model with the superior recall score. This systematic approach aimed to prioritize models that not only excelled in correctly classifying the minority class (ensuring high specificity) but also maintained a commendable recall, which is vital for comprehensively capturing instances belonging to the minority class.

2.1.3 Results

Next, we display tables showcasing the various outcomes:

Architecture	Accuracy	Recall	Specificity	F1-Score
20	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*
40	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*
80	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*
100	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*
60, 120	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*
80, 50	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*
80, 100	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*
100, 40	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*

Table 1: ANN metrics result for CrossValidation

Architecture	Accuracy	Recall	Specificity	F1-Score	CM
20	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]
40	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]
80	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]
100	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]
60, 120	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]
80, 50	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]
80, 100	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]
100, 40	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]

Table 2: ANN metrics result

k	Accuracy	Recall	Specificity	F1-Score
1	88.08% *(0.01)*	93.17% *(0.01)*	12.29% *(0.08)*	93.61% *(0.01)*
2	83.3% *(0.01)*	87.66% *(0.02)*	18.43% *(0.11)*	90.77% *(0.01)*
3	92.72% *(0.01)*	98.84% *(0.01)*	1.7% *(0.03)*	96.22% *(0.0)*
5	93.5% *(0.0)*	99.74% *(0.0)*	0.56% *(0.02)*	96.64% *(0.0)*
7	93.6% *(0.0)*	99.89% *(0.0)*	0.0% *(0.0)*	96.7% *(0.0)*
10	93.6% *(0.0)*	99.89% *(0.0)*	0.0% *(0.0)*	96.7% *(0.0)*
15	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*

Table 3: kNN metrics result for CrossValidation

k	Accuracy	Recall	Specificity	F1-Score	CM
1	87.22%	94.19%	8.62%	93.12%	[616 53; 38 5]
2	80.48%	86.39%	13.79%	89.05%	[565 50; 89 8]
3	90.03%	97.86%	1.72%	94.74%	[640 57; 14 1]
5	91.43%	99.54%	0.0%	95.52%	[651 58; 3 0]
7	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]
10	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]
15	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]

Table 4: kNN metrics result

MaxDepth	Accuracy	Recall	Specificity	F1-Score
3	93.6% *(0.0)*	99.89% *(0.0)*	0.0% *(0.0)*	96.7% *(0.0)*
5	93.15% *(0.0)*	99.36% *(0.0)*	0.56% *(0.02)*	96.45% *(0.0)*
7	92.34% *(0.0)*	98.39% *(0.0)*	2.22% *(0.03)*	96.01% *(0.0)*
10	91.42% *(0.01)*	97.22% *(0.01)*	5.03% *(0.04)*	95.5% *(0.01)*
15	88.96% *(0.01)*	94.26% *(0.02)*	10.03% *(0.06)*	94.11% *(0.01)*
nothing	87.1% *(0.02)*	92.08% *(0.02)*	12.88% *(0.08)*	93.04% *(0.01)*

Table 5: DecisionTree metrics result for CrossValidation

MaxDepth	Accuracy	Recall	Specificity	F1-Score	CM
3	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]
5	91.71%	99.85%	0.0%	95.68%	[653 58; 1 0]
7	89.61%	97.55%	0.0%	94.52%	[638 58; 16 0]
10	88.06%	95.72%	1.72%	93.64%	[626 57; 28 1]
15	85.25%	92.66%	1.72%	92.03%	[606 57; 48 1]
nothing	84.55%	91.9%	1.72%	91.62%	[601 57; 53 1]

Table 6: DecisionTree metrics result

Kernel	C	Accuracy	Recall	Specificity	F1-Score
rbf	0.1	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*
rbf	1.0	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*
rbf	10.0	93.6% *(0.0)*	99.89% *(0.0)*	0.0% *(0.0)*	96.7% *(0.0)*
poly	0.1	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*
poly	1.0	93.67% *(0.0)*	99.96% *(0.0)*	0.0% *(0.0)*	96.73% *(0.0)*
linear	0.1	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*
linear	1.0	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*
linear	10.0	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*

Table 7: SVM metrics result for CrossValidation

Kernel	C	Accuracy	Recall	Specificity	F1-Score	CM
rbf	0.1	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]
rbf	1.0	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]
rbf	10.0	91.71%	99.85%	0.0%	95.68%	[653 58; 1 0]
poly	0.1	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]
poly	1.0	91.71%	99.85%	0.0%	95.68%	[653 58; 1 0]
linear	0.1	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]
linear	1.0	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]
linear	10.0	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]

Table 8: SVM metrics result

Ensemble Model	Parameters	Accuracy	Recall	Specificity	F1-Score
Voting Hard		92.93% *(0.0)*	99.06% *(0.01)*	1.7% *(0.03)*	96.33% *(0.0)*
Stacking (DecisionTree)	Max Depth 10	92.06% *(0.01)*	98.16% *(0.02)*	1.11% *(0.02)*	95.86% *(0.01)*
Stacking (kNN)	numNeighboors = 2	82.88% *(0.04)*	87.96% *(0.04)*	7.32% *(0.09)*	90.54% *(0.03)*
Stacking (SVM)	C = 0.1, kernel = rbf	93.71% *(0.0)*	100.0% *(0.0)*	0.0% *(0.0)*	96.75% *(0.0)*

Table 9: Ensemble Model metrics result for CrossValidation

Ensemble Model	Final Estimator	Accuracy	Recall	Specificity	F1-Score	CM
Voting Hard		91.15%	99.08%	1.72%	95.36%	[648 57; 6 1]
Stacking (DecisionTree)	Max Depth 10	90.59%	98.62%	0.0%	95.06%	[645 58; 9 0]
Stacking (kNN)	numNeighboors = 2	81.18%	87.16%	13.79%	89.48%	[570 50; 84 8]
Stacking (SVM)	C = 0.1, kernel = rbf	91.85%	100.0%	0.0%	95.75%	[654 58; 0 0]

Table 10: Ensemble Model metrics result

In the initial approach, the imbalanced nature of the dataset heavily influenced the obtained results, with the benign class accounting for only 7%. Although the models achieved nearly 100% recall for malware detection, the specificity dropped to 0%, failing to identify any instances of the benign class. The reported accuracy of approximately 91% holds little substantive value in this context, emphasizing the misleading nature of accuracy metrics when dealing with imbalanced datasets. These outcomes highlight the importance of using alternative evaluation met-

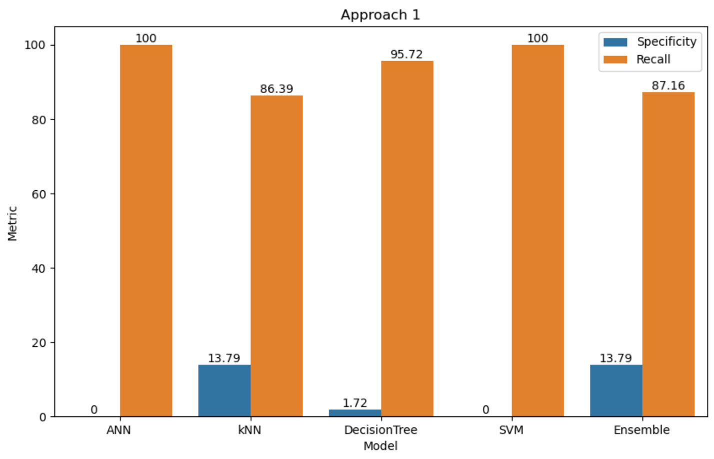

Figure 3: Performance Evaluation: Contrasting Recall and Specificity among Various Models

rics beyond accuracy to accurately assess model performance in imbalanced scenarios.

Artificial Neural Networks When exploring different network topologies, such as feature funnels or expansions, it was attempted to modify the flow and complexity of information processing within the neural network architectures. Feature funnels gradually reduce network width to distill essential features, while expansions widen the network to enable more intricate information processing and feature representation. Although structural modifications were made to alter information flow and feature extraction, the ANN models still showed a consistent trade-off between specificity and recall, ranging from 0 to 100 (showed in figures 1 and 2). This trend highlights the model's tendency to label all instances as malware, indicating a need for further investigation or adjustments to address this inherent bias.

k-Nearest Neighbors The K-Nearest Neighbors (KNN) model was trained using seven different values of k (1, 2, 3, 5, 7, 10, 15). Although configuring with even numbers is typically discouraged, an exploration into the model's performance on untie votes (by distance, as documented in sklearn [5]) was conducted.

The resulting tables 3 and 4 reveal intriguing insights: lower k values showcase superior performance in the recall-specificity trade-off. For instance, $k = 3$ demonstrates a f1-score of 96.22%, a high recall of 97.86%, yet exhibits a specificity of merely 1.72%. The favorable performance of lower k values can be attributed to their proximity-based classification, which relies on fewer neighboring data points for decision-making. However, as k increases beyond a certain threshold, the model tends to generalize more, resulting in best unbalanced metrics like accuracy.

Also, in the evaluation, $k = 2$ demonstrated notable performance metrics, despite being an even number. With a respectable recall of 86.39%, and a comparatively higher specificity of 13.79%, $k = 2$ presented an intriguing deviation from the trend observed in other even k values. This anomaly suggests that $k = 2$ managed to strike a balance between considering neighboring points while avoiding excessive bias from a single nearest neighbor.

Decision Tree The Decision Tree model was tested with varying depths (3, 5, 7, 10, 15 and unlimited), but no significant changes were observed in its performance. However, in the final test results (table 6), a clear trend emerged: from a depth of 10 down to no specified depth, there was a slight increase in specificity, rising to 1.72% from a previous value of 0%.

Additionally, during the cross-validation

[5]Link to KNeighborsClassifier documentation by Scikit Learn

phase (table 5), the specificity consistently increased, peaking at 12.88%. The data shows that as the tree depth increases, there is a consistent pattern of specificity growth, with a minor standard deviation of 0.08. This suggests a positive correlation between the maximum depth of the decision tree and its specificity, indicating that expanding the tree depth could improve model performance.

Support Vector Machine The Support Vector Machines (SVMs) underwent testing with various kernel and C parameter combinations, including `rbf` with $C = 0.1, 1.0, 10.0$, `poly` with $C = 0.1, 1.0$ and `linear` with $C = 0.1, 1.0, 10.0$.

However, the obtained results (table 8 and 7) closely mirrored those of the Artificial Neural Networks (ANNs). The evaluations showcased a recurrent trade-off between recall and specificity, resulting in a consistent 100-0 pattern where all patterns were classified as malware. Only in a couple of observed parameters within the confusion matrix, there were instances where a malware pattern was incorrectly classified as benign, causing a drop in recall from 100. Overall, these SVM models exhibited a consistent bias towards classifying all instances as malware, emphasizing the need for further exploration or adjustment to mitigate this inherent bias and enhance model performance.

Ensembles The ensemble models were trained using four different configurations: Hard Voting, Stacking with Decision Tree ($MaxDepth = 10$), Stacking with kNN ($numNeighboors = 2$), and Stacking with SVM ($C = 0.1, kernel = rbf$). Each configuration showcased distinct trade-offs in performance metrics.

Hard Voting and Stacking with SVM exhibited relatively similar performances in both training and cross-validation (tables 9 and 10), with high accuracy and recall but notably lower specificity (along the trend seen). Stacking with Decision Tree showcased consistent metrics, albeit with lower specificity. Stacking with kNN revealed the largest variability across its metrics.

During cross-validation, these models displayed consistent tendencies in their metrics, maintaining the patterns observed during training. The high recall rates across all models were notable, yet were often accompanied by compromised specificity. Stacking with kNN displayed larger fluctuations in its performance metrics, indicating its susceptibility to variations across validation folds.

While these ensemble models exhibit promising high recall rates, indicating their proficiency in identifying malware instances, there remains the consistent challenge with specificity due to the class imbalance. This suggests a potential bias towards categorizing most instances as malware.

Final result of the approach In summary, this approach produces challenging results, particularly reflected in the specificity metrics across all models. The pervasive trend observed in these models is a tendency to classify most instances, including those from the benign class, as malware. Consequently, the primary challenge is to address this bias by effectively balancing the benign class to achieve more accurate and balanced classifications. Achieving a more refined balance between classes is critical to improving the models' ability to discriminate between benign and malware instances, and ultimately improving overall classification performance.

2.2 Approach 2: Binary Classification with Balanced Classes

In this second approximation phase, it was decided to continue with binary classification in order to address the class discrepancy in the dataset. With only 7% of the instances belonging to the "benign" class, the need to apply an oversampling strategy specifically to this category was identified. The basic purpose is to balance the representation of this class in relation to the malware class present in the dataset. For this purpose, the `MLDataPattern`[6] in `Julia` provides specialised methods to manipulate and modify unbalanced data. This strategy is essential in order to increase the representation of the minority class, thus attempting to avoid potential biases that could arise in the resulting model due to the discrepancy in class distribution.

[6]Link to Package Documentation package

Label	Entries
Benign	3,320
Malware	3,320
Total size	6,640

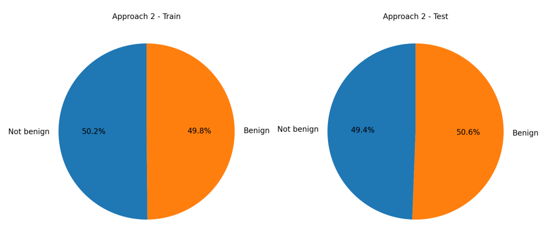

Figure 4: Label distribution (second approach)

With the implementation of this data balancing strategy, a noticeable change in the distribution of class labels has been generated. Contrary to the previous structure (shown in table 2), the number of instances between the two classes has been balanced and an equal distribution has been achieved since the table 4 shows. This change was fundamental, as the minority class, which had previously been much less represented, was brought up to the same level as the majority class. This adjustment of the class distribution is of paramount importance, as it promotes an environment conducive to training the model, reducing possible biases and allowing better generalisation of the patterns present in the data.

2.2.1 Data Preprocessing

The data processing phase followed the methodology used in the first approach, retaining the steps of cleaning and feature selection. However, two crucial components was added: the incorporation of data balancing and the standardization.

The process of balancing the class distribution played a pivotal role in rectifying the inherent imbalances present within the dataset. This corrective measure focused particularly on mitigating the disproportionate representation among different classes, primarily by implementing oversampling techniques tailored to address the underrepresented minority class. By strategically augmenting the instances within the minority class, the aim was to achieve a more equitable distribution across all classes. Oversampling served as a critical strategy, amplifying the instances of the minority class to align more closely with the frequency of the majority classes. This concerted effort in rebalancing the dataset ensures that the model isn't biased toward the overrepresented classes, thereby enhancing its ability to learn and generalize patterns equally across all classes during the training phase.

About the standardization, method used in scaling data, was chosen over Min-Max scaling due to its robustness in handling outliers or extreme values commonly present in datasets. Min-Max scaling confines values within a specified range (normally 0-1), which can be problematic when dealing with outliers as it distorts the overall range of values. In contrast, standardization rescales data to have a mean of zero and a standard deviation of one. This method is resilient against outliers because it centers the data around its mean and measures dispersion in terms of standard deviations. Unlike Min-Max scaling, standardization doesn't confine values to a specific range, preserving the relative relationships and distributions among features while ensuring that outliers or unusual values don't unduly influence the model's learning process. Hence, in datasets with potential anomalous or peak values, standardization stands out as a more robust and suitable scaling method, maintaining the integrity of the data's distribution and characteristics.

2.2.2 Other data related to the experiment

Cross Validation In alignment with the methodology adopted in the initial approach, the data handling process for this approach remained consistent. A 10-fold cross-validation technique was employed for robust model evaluation, ensuring comprehensive validation across various subsets of the dataset.

Train-test split A 20% data split was reserved for testing purposes using the hold-out method, maintaining consistency in the assessment approach.

Feature engineering Similarly, the feature selection and engineering methodologies mirrored those implemented previously, involving the elimination of columns containing negligible information (i.e., only a single value). Moreover, a transformation was applied to IP address strings, converting them into IP octets, thereby enhancing the interpretability and utility of the data. Post this process, the dataset retained 70 columns, representing the refined and pertinent attributes for subsequent model training and evaluation.

ANN hyperparameters Additionally, it's important to note that the hyperparameters for the Artificial Neural Network (ANN) model remained unchanged, following the same configuration as in the initial approach, ensuring consistency in the modeling framework across experiments.

Model selection for ensembles criteria In the second approach, where class balance was achieved, the focus shifted during ensemble model selection towards both recall and specificity metrics, while emphasizing a balanced consideration between these evaluation measures. Unlike the previous approach, where specificity held primary importance due to class imbalance, this time, a more nuanced approach was adopted. The emphasis was on achieving a balanced tradeoff between recall and specificity, ensuring that the model's performance was not skewed towards exceptionally high scores in either metric at the expense of the other. The objective was to prioritize models demonstrating a harmonious balance between recall and specificity, acknowledging that a balanced performance across these metrics is crucial for the overall effectiveness of the model in capturing both positive and negative instances without favoring one over the other.

2.2.3 Results

Next, we will present tables that display various outcomes:

Architecture	Accuracy	Recall	Specificity	F1-Score
20	60.88% *(0.02)*	58.13% *(0.02)*	63.63% *(0.04)*	59.48% *(0.01)*
40	59.06% *(0.01)*	56.72% *(0.03)*	61.41% *(0.05)*	56.85% *(0.02)*
80	55.24% *(0.01)*	56.49% *(0.04)*	53.99% *(0.04)*	52.61% *(0.02)*
100	54.47% *(0.01)*	52.87% *(0.04)*	56.06% *(0.05)*	49.85% *(0.02)*
60, 120	53.94% *(0.01)*	59.53% *(0.08)*	48.36% *(0.09)*	49.12% *(0.04)*
80, 50	56.5% *(0.01)*	56.82% *(0.04)*	56.19% *(0.04)*	52.23% *(0.01)*
80, 100	54.67% *(0.01)*	55.69% *(0.05)*	53.65% *(0.06)*	47.75% *(0.03)*
100, 40	56.83% *(0.01)*	59.17% *(0.06)*	54.49% *(0.08)*	53.57% *(0.03)*

Table 11: ANN metrics result for CrossValidation

Architecture	Accuracy	Recall	Specificity	F1-Score	CM
20	59.64%	64.31%	54.97%	61.44%	[427 299; 237 365]
40	57.23%	77.41%	37.05%	64.41%	[514 418; 150 246]
80	54.29%	72.74%	35.84%	61.41%	[483 426; 181 238]
100	52.79%	62.5%	43.07%	56.97%	[415 378; 249 286]
60, 120	54.44%	16.27%	92.62%	26.31%	[108 49; 556 615]
80, 50	53.77%	11.14%	96.39%	19.42%	[74 24; 590 640]
80, 100	56.55%	25.0%	88.1%	36.52%	[166 79; 498 585]
100, 40	58.89%	83.89%	33.89%	67.11%	[557 439; 107 225]

Table 12: ANN metrics result

k	Accuracy	Recall	Specificity	F1-Score
1	95.33% *(0.01)*	90.66% *(0.02)*	100.0% *(0.0)*	95.1% *(0.01)*
2	91.3% *(0.01)*	82.6% *(0.02)*	100.0% *(0.0)*	90.46% *(0.01)*
3	91.3% *(0.01)*	82.6% *(0.02)*	100.0% *(0.0)*	90.46% *(0.01)*
5	87.71% *(0.01)*	75.41% *(0.02)*	100.0% *(0.0)*	85.97% *(0.01)*
7	83.98% *(0.01)*	67.96% *(0.03)*	100.0% *(0.0)*	80.89% *(0.02)*
10	77.97% *(0.01)*	56.06% *(0.03)*	99.89% *(0.0)*	71.76% *(0.02)*
15	72.36% *(0.01)*	48.87% *(0.02)*	95.86% *(0.02)*	63.86% *(0.02)*

Table 13: kNN metrics result for CrossValidation

k	Accuracy	Recall	Specificity	F1-Score	CM
1	96.01%	92.02%	100.0%	95.84%	[611 0; 53 664]
2	91.49%	82.98%	100.0%	90.7%	[551 0; 113 664]
3	91.49%	82.98%	100.0%	90.7%	[551 0; 113 664]
5	87.73%	75.45%	100.0%	86.01%	[501 0; 163 664]
7	85.02%	70.03%	100.0%	82.37%	[465 0; 199 664]
10	80.12%	60.24%	100.0%	75.19%	[400 0; 264 664]
15	74.7%	50.45%	98.95%	66.6%	[335 7; 329 657]

Table 14: kNN metrics result

MaxDepth	Accuracy	Recall	Specificity	F1-Score
3	56.85% *(0.02)*	42.33% *(0.22)*	71.37% *(0.22)*	46.65% *(0.13)*
5	61.8% *(0.02)*	42.16% *(0.12)*	81.44% *(0.14)*	51.67% *(0.06)*
7	67.28% *(0.02)*	51.42% *(0.13)*	83.14% *(0.09)*	60.14% *(0.08)*
10	77.59% *(0.03)*	64.57% *(0.08)*	90.62% *(0.05)*	74.01% *(0.05)*
15	88.57% *(0.02)*	78.77% *(0.04)*	98.38% *(0.01)*	87.28% *(0.03)*
None	95.29% *(0.01)*	90.59% *(0.02)*	100.0% *(0.0)*	95.05% *(0.01)*

Table 15: DecisionTree metrics result for CrossValidation

MaxDepth	Accuracy	Recall	Specificity	F1-Score	CM
3	58.89%	43.52%	74.25%	51.42%	[289 171; 375 493]
5	63.03%	49.55%	76.51%	57.27%	[329 156; 335 508]
7	71.23%	58.73%	83.73%	67.13%	[390 108; 274 556]
10	81.4%	65.81%	96.99%	77.97%	[437 20; 227 644]
15	91.04%	82.08%	100.0%	90.16%	[545 0; 119 664]
None	96.31%	92.62%	100.0%	96.17%	[615 0; 49 664]

Table 16: DecisionTree metrics result

Kernel	C	Accuracy	Recall	Specificity	F1-Score
rbf	0.1	57.27% *(0.03)*	62.57% *(0.03)*	51.96% *(0.06)*	59.43% *(0.02)*
rbf	1.0	64.19% *(0.03)*	58.09% *(0.04)*	70.3% *(0.05)*	61.83% *(0.03)*
rbf	10.0	71.74% *(0.02)*	64.34% *(0.04)*	79.14% *(0.04)*	69.46% *(0.03)*
poly	0.1	57.46% *(0.02)*	90.14% *(0.02)*	24.78% *(0.05)*	67.95% *(0.01)*
poly	1.0	62.07% *(0.03)*	77.63% *(0.03)*	46.5% *(0.06)*	67.19% *(0.02)*
linear	0.1	57.32% *(0.01)*	56.81% *(0.05)*	57.83% *(0.06)*	57.0% *(0.02)*
linear	1.0	58.74% *(0.01)*	60.13% *(0.05)*	57.35% *(0.06)*	59.23% *(0.02)*
linear	10.0	59.32% *(0.02)*	62.72% *(0.04)*	55.92% *(0.06)*	60.63% *(0.02)*

Table 17: SVM metrics result for CrossValidation

Kernel	C	Accuracy	Recall	Specificity	F1-Score	CM
rbf	0.1	57.3%	59.49%	55.12%	58.22%	[395 298; 269 366]
rbf	1.0	63.18%	56.63%	69.73%	60.6%	[376 201; 288 463]
rbf	10.0	72.44%	61.75%	83.13%	69.14%	[410 112; 254 552]
poly	0.1	56.78%	89.31%	24.25%	67.39%	[593 503; 71 161]
poly	1.0	63.93%	76.96%	50.9%	68.09%	[511 326; 153 338]
linear	0.1	58.51%	54.52%	62.5%	56.78%	[362 249; 302 415]
linear	1.0	58.66%	57.08%	60.24%	58.0%	[379 264; 285 400]
linear	10.0	59.19%	60.54%	57.83%	59.73%	[402 280; 262 384]

Table 18: SVM metrics result

Ensemble Model	Parameters	Accuracy	Recall	Specificity	F1-Score
Hard Voting		94.33% *(0.01)*	88.67% *(0.02)*	100.0% *(0.0)*	93.98% *(0.01)*
Stacking (Decision Tree)	Max Depth = None	97.27% *(0.01)*	98.42% *(0.01)*	96.12% *(0.02)*	97.31% *(0.01)*
Stacking (kNN)	numNeighboors = 1	97.27% *(0.01)*	98.42% *(0.01)*	96.12% *(0.02)*	97.31% *(0.01)*
Stacking (SVM)	C = 10, kernel = rbf	99.1% *(0.0)*	98.19% *(0.01)*	100.0% *(0.0)*	99.09% *(0.0)*

Table 19: Ensemble Model metrics result for CrossValidation

Ensemble Model	Final Estimator	Accuracy	Recall	Specificity	F1-Score	CM
Hard Voting		95.11%	90.21%	100.0%	94.85%	[599 0; 65 664]
Stacking (Decision Tree)	Max Depth = None	95.26%	98.34%	92.17%	95.4%	[653 52; 11 612]
Stacking (kNN)	numNeighboors = 1	95.26%	98.34%	92.17%	95.4%	[653 52; 11 612]
Stacking (SVM)	C = 10, kernel = rbf	99.02%	98.04%	100.0%	99.01%	[651 0; 13 664]

Table 20: Ensemble Model metrics result

In this approach, it's important to note that the parameters for all models remain consistent with the previous approach, except for the ensembles. The ensemble configurations have been adjusted based on the best performing models within this balanced dataset.

Artificial Neural Networks In the second approach, achieving improved specificity indicated the effectiveness of data balancing techniques. With a balanced dataset, the importance of accuracy as a metric became more pronounced. However, observed accuracy values only marginally

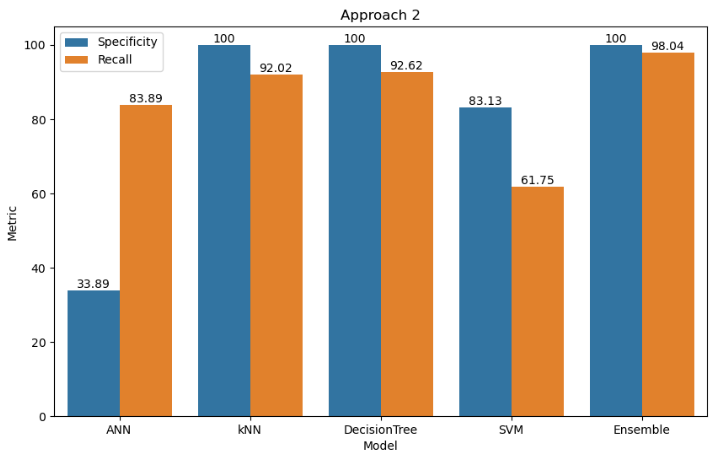

Figure 5: Performance Evaluation: Contrasting Recall and Specificity among Various Models

surpassed the 50% threshold, suggesting a modest enhancement over random chance (where each class has a 50% probability). Notably, two distinct trends emerged: one showcased a balanced recall and specificity, approximately 50% or 60%-40%, while the other exhibited marked imbalances, such as low recall and high specificity and vice versa..

k-Nearest Neighbors The performance of the KNN model across different architectures shows remarkable trends. As model complexity increases, there's a consistent decrease in accuracy (table 14), particularly noticeable from $k = 1$ to $k = 15$. This suggests that overly complex models may struggle with generalisation and may overfit the data.

There's also a steady decline in recall (malware detection) with increasing complexity. This decrease suggests that more complex models may miss important patterns that are critical for identifying malware instances. Despite this, specificity remains consistently high across all architectures, demonstrating the model's ability to accurately detect benign instances.

The F1 score, which combines precision and recall, shows a similar decreasing trend with increasing model complexity. This decrease suggests a trade-off in the model's ability to balance precision in identifying benign instances with recall in detecting malware.

The cross-validation results (table 13) confirm these observations, showing a parallel decrease in precision, recall and F1 score as model complexity increases. However, the standard deviations associated with these metrics indicate some variability in performance across folds, which is more pronounced for complex architectures.

Decision Trees The Decision Tree Classifier shows a significant improvement in performance with increasing maximum depth. In the cross-validation (table 15), as the maximum depth grows, there's a consistent rise in accuracy, recall, specificity, and F1-score. This indicates that deeper trees capture more intricate patterns, enhancing both malware detection and benign instance recognition.

Looking at the final results (table 16), a similar trend emerges. As the maximum depth increases from 3 to None (no depth limit), there's a steady improvement across all metrics: accuracy, recall, specificity, and F1-score. Notably, at maximum depth (None), the model achieves the highest performance, with accuracy and recall exceeding 96% and specificity at 100%. This indicates a robust ability to detect both malware and benign instances accurately.

The confusion matrices also highlight this enhancement, with fewer misclassifications as the tree depth increases. At maximum depth, the model shows minimal false negatives and false positives, signifying a high precision in both classifying malware and benign instances.

The Decision Tree Classifier demonstrates a clear advantage with deeper trees, showcasing improved performance metrics and better generalization, particularly evident in accurately detecting malware instances while maintaining high specificity in recognizing benign instances. With this, it can be said that a deeper decision tree tends to yield better accuracy as it captures more intricate patterns in the data, leading to improved overall performance.

Support Vector Machines The SVM models with different kernels (`rbf, poly, linear`) and various values of the hyperparameter C showcase distinct behaviors.

For the `rbf` kernel, increasing the regularization parameter `C` from 0.1 to 10.0 leads to a noticeable enhancement in accuracy, recall, specificity, and F1-score. Higher `C` values contribute to better overall performance, as observed in the cross-validation results and final evaluation metrics (tables 17 and 18). This signifies that a stronger regularization (higher `C`) allows the model to capture more complex relationships within the data, leading to improved classification.

In contrast, the `poly` kernel demonstrates a sensitivity to the choice of `C`. Lower values of `C` (0.1) result in a significant trade-off between recall and specificity, highlighting overfitting tendencies with very high recall but low specificity. However, increasing `C` to 1.0 showcases an improvement in specificity while maintaining a reasonable balance between recall and specificity. This suggests a better generalization capacity with moderate `C` values for the `poly` kernel.

The `linear` kernel demonstrates a relatively consistent but moderate performance across different `C` values. There's a subtle increase in performance metrics with higher `C`, indicating a minor enhancement in accuracy and recall, though the overall improvement remains modest compared to the `rbf` and `poly` kernels.

The `rbf` kernel demonstrates robust performance with higher `C` values, showcasing improved accuracy, recall, and specificity and demonstrating that it's one of the most advanced (in some problems) kernels as it can handle non-linear relationships between features more effectively. The `poly` kernel requires careful tuning of `C` to balance between recall and specificity, while the `linear` kernel showcases more stability but limited improvements with higher `C` as it's a simpler kernel. The choice of kernel and `C` values significantly influences the SVM's performance, with `rbf` and `poly` kernels being more sensitive to `C` adjustments.

Ensembles The ensemble models showcase a combination of individual classifiers, demonstrating robust performance and significant improvements compared to individual base models. The base models selected are Decision Tree (`MaxDepth = None`), kNN (`k neighbors = 1`) and SVM (`Kernel rbf, C = 10`).

The cross validation results (table 19) shows that the standard desviations are quite low wich means consistency between folds, asserting the ensemble don't have high variance.

About the final results (table 20), the hard voting ensemble achieves an impressive overall accuracy of 95.11%, with a balanced specificity of 100% and a strong recall of 90.21%. This indicates a reliable ability to correctly classify both malware and benign instances.

The stacking models, utilizing decision tree, kNN, and SVM as base estimators, consistently perform at a high level. They showcase matching results in terms of accuracy, recall, specificity, and F1-score, reaching more than 95% accuracy. Notably, all three stacking models achieve exceptionally high recall (above 98%) with varying high specificity, showcasing their ability to effectively detect malware instances while maintaining a low rate of false negatives.

The confusion matrices for these ensembles highlight their efficiency in minimizing misclassifications, with notably low false negatives across all final ensemble models. This indicates a high precision in identifying both malware and benign instances, crucial in security-related tasks like malware detection.

The ensemble models, whether using hard voting or stacking methodologies, demonstrate substantial enhancements over individual base models, showcasing superior accuracy, recall, and specificity, crucial for robust malware detection systems. In fact, the ensemble model with `svm` as final estimator, showcases a almost perfect classification, only failing 13 classifications of the malware class, which shows a substantial improvement.

Final result of the approach The comparison between the first and the refined approach shows a significant improvement in the correct classification of benign instances, highlighting the crucial impact of data set balancing in achieving improved metrics. Finally, the most notable results are achieved by the ensemble models, further validating their capabilities as a powerful technique in machine learning, demonstrating exceptional accuracy and adeptness in classifying both malware and benign instances.

Despite the use of various configurations such as funnels or extensions, Artificial Neural Networks (ANNs) failed to demonstrate competitive performance when compared to other machine learning models. This underperformance of ANNs further highlights the superiority of alternative models in this particular classification task, highlighting their ability to outperform ANNs in accurately classifying malware and benign instances. The comparative underperformance of ANNs emphasises the need for diverse model experimentation and the recognition that different machine learning approaches may excel in different scenarios.

2.3 Approach 3: Multiclass Classification

In this third approach, a shift towards multiclass classification has been adopted, aiming to classify various types of Android malware along with the benign instances. Learning from the lessons of prior approaches, where class imbalances posed challenges, a pivotal focus has been placed on achieving class balance. Given the disparity among classes, particularly with the minority class, the emphasis lies in balancing the dataset to enhance the model's accuracy and effectiveness. Recognizing the intricacies of dealing with imbalanced classes, the primary strategy involves implementing techniques for data balancing. Addressing this imbalance is crucial for the model to effectively learn from the minority class, thereby improving its ability to accurately classify different types of Android malware.

In Table 6, the distribution illustrates a balanced representation across the four classes, each containing precisely 1,475 instances. This balanced distribution ensures an equal footing for all classes within the dataset, facilitating a more equitable learning environment for the classification model.

Label	Entries
Benign	1,475
Android_Scareware	1,475
Android_SMS_Malware	1,475
Android_Adware	1,475
Total size	5,900

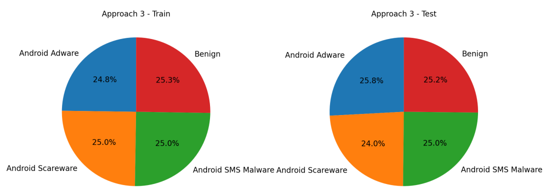

Figure 6: Label distribution (third approach)

2.3.1 Data Preprocessing

Continuing from the learning acquired in the second approach, where the problem was binary in nature, several principles persisted in the third approach, now addressing a multiclass problem. Among these, the foundational practice of ensuring class balance was upheld, maintaining a proportional representation across all classes within the dataset. Additionally, the decision to prioritize standardization over scaling methods like MinMax scaling remained consistent. Unlike the MinMax scaling, standardization, which centers the data around zero with a standard deviation of one, was preferred due to its robustness in handling outliers and its ability to maintain the data's inherent distribution without restricting values to a specific range. These established methodologies, carried forward from the binary problem to the multiclass setting, underscore the importance of maintaining balanced class representation and employing robust data preprocessing techniques tailored to the nature of the dataset.

2.3.2 Other data related to the experiment

Cross Validation Consistency prevailed in the data handling process for the third approach, mirroring the methodology implemented in the initial stages. Robust model evaluation persisted through the application of a 10-fold cross-validation technique, ensuring thorough validation across diverse subsets of the dataset. This approach maintained a steadfast commitment to comprehensive validation, enabling a thorough assessment of model performance and generalizability across different sections of the dataset.

Train-test split A 20% data split was reserved for testing purposes using the hold-out method, maintaining consistency in the assessment approach.

Feature engineering Likewise, the third approach adhered to analogous feature selection and engineering strategies observed in preceding iterations. This involved the systematic removal of columns featuring insignificant information, specifically those containing only a single value. Additionally, an IP address transformation was executed, converting the strings into IP octets to amplify the data's interpretability and functional value. Following these refinements, the dataset was streamlined to retain 70 columns, signifying the culmination of a refined set of pertinent attributes essential for subsequent stages of model training and evaluation.

ANN hyperparameters Furthermore, consistency prevailed in maintaining the hyperparameters for the Artificial Neural Network (ANN) model throughout the third approach. These parameters remained unaltered, preserving the same configuration as employed in the initial approach. This deliberate decision aimed to ensure a consistent and standardized modeling framework across all experiments, facilitating a direct comparison of model performances and outcomes between different approaches.

Model selection for ensembles criteria Given the transition to addressing a multiclass problem, a shift in the criteria for model selection within ensembles was enacted. The focus now shifted towards evaluating models based on accuracy and F1 score metrics. These metrics were employed to meticulously assess and determine the best hyperparameters for each model, facilitating their inclusion within the ensembles. This shift in approach was essential to identify the most proficient models adept at achieving high accuracy in multiclass classification while also considering the F1 score, which encapsulates a balance between precision and recall for each class. The emphasis on accuracy and F1 score aimed to ensure the selection of models that not only perform well overall but also exhibit commendable performance across individual classes within the multiclass setting.

2.3.3 Results

A set of tables demonstrating different results will be shown next.

Architecture	Accuracy	Recall	Specificity	F1-Score
20	33.21% *(0.01)*	33.21% *(0.01)*	77.75% *(0.0)*	32.2% *(0.01)*
40	32.82% *(0.01)*	32.82% *(0.01)*	77.61% *(0.0)*	30.85% *(0.01)*
80	31.43% *(0.01)*	31.43% *(0.01)*	77.13% *(0.0)*	27.34% *(0.01)*
100	30.83% *(0.01)*	30.83% *(0.01)*	76.95% *(0.0)*	26.02% *(0.01)*
60, 120	30.2% *(0.0)*	30.2% *(0.0)*	76.71% *(0.0)*	21.09% *(0.01)*
80, 50	31.67% *(0.01)*	31.67% *(0.01)*	77.22% *(0.0)*	25.56% *(0.01)*
80, 100	30.12% *(0.01)*	30.12% *(0.01)*	76.71% *(0.0)*	21.38% *(0.01)*
100, 40	31.61% *(0.01)*	31.61% *(0.01)*	77.21% *(0.0)*	25.89% *(0.01)*

Table 21: ANN metrics result for CrossValidation

Architecture	Accuracy	Recall	Specificity	F1-Score	CM
20	22.37%	22.37%	73.78%	20.66%	[67 104 105 26; 78 56 135 35; 76 64 124 32; 117 48 96 17]
40	20.59%	20.59%	72.99%	17.85%	[147 80 55 20; 166 35 68 35; 171 61 43 21; 175 48 37 18]
80	23.9%	23.9%	74.66%	23.48%	[78 87 51 86; 86 86 36 96; 69 97 41 89; 87 88 26 77]
100	21.44%	21.44%	73.11%	17.1%	[74 200 15 13; 105 155 12 32; 107 153 9 27; 128 120 15 15]
60, 120	23.64%	23.64%	75.28%	17.65%	[3 44 132 123; 7 12 137 148; 8 21 115 152; 8 18 103 149]
80, 50	21.44%	21.44%	74.18%	16.98%	[12 169 20 101; 28 99 10 167; 22 115 10 149; 31 101 14 132]
80, 100	25.42%	25.42%	74.6%	17.27%	[239 24 31 8; 208 7 65 24; 220 12 45 19; 219 7 43 9]
100, 40	23.9%	23.9%	75.43%	17.48%	[6 29 126 141; 12 5 139 148; 9 15 119 153; 20 8 98 152]

Table 22: ANN metrics result

k	Accuracy	Recall	Specificity	F1-Score
1	71.65% *(0.02)*	71.65% *(0.02)*	90.54% *(0.01)*	70.28% *(0.02)*
2	62.31% *(0.02)*	62.31% *(0.02)*	87.44% *(0.01)*	60.55% *(0.02)*
3	57.35% *(0.02)*	57.35% *(0.02)*	85.78% *(0.01)*	55.48% *(0.02)*
5	52.04% *(0.02)*	52.04% *(0.02)*	84.0% *(0.01)*	49.75% *(0.02)*
7	47.67% *(0.03)*	47.67% *(0.03)*	82.57% *(0.01)*	45.61% *(0.03)*
10	41.42% *(0.03)*	41.42% *(0.03)*	80.48% *(0.01)*	40.28% *(0.03)*
15	39.47% *(0.03)*	39.47% *(0.03)*	79.83% *(0.01)*	38.99% *(0.03)*

Table 23: kNN metrics result for CrossValidation

k	Accuracy	Recall	Specificity	F1-Score	CM
1	71.78%	71.78%	90.64%	70.38%	[302 0 0 0; 22 143 85 54; 14 86 154 42; 4 10 16 248]
2	63.73%	63.73%	87.91%	61.89%	[297 5 0 0; 22 172 45 65; 14 133 82 67; 4 67 6 201]
3	57.97%	57.97%	86.01%	56.11%	[297 5 0 0; 35 127 70 72; 43 85 100 68; 17 62 39 160]
5	56.19%	56.19%	85.36%	54.22%	[285 5 8 4; 46 128 68 62; 65 72 100 59; 35 53 40 150]
7	50.42%	50.42%	83.37%	48.24%	[270 14 8 10; 67 112 73 52; 81 72 87 56; 48 54 50 126]
10	43.98%	43.98%	81.21%	42.53%	[216 30 28 28; 84 109 60 51; 91 72 74 59; 52 58 48 120]
15	40.0%	40.0%	79.94%	39.13%	[182 36 46 38; 78 100 72 54; 81 70 75 70; 66 49 48 115]

Table 24: kNN metrics result

MaxDepth	Accuracy	Recall	Specificity	F1-Score
3	33.41% *(0.02)*	33.41% *(0.02)*	77.87% *(0.01)*	29.11% *(0.03)*
5	36.65% *(0.03)*	36.65% *(0.03)*	78.93% *(0.01)*	34.46% *(0.03)*
7	40.68% *(0.02)*	40.68% *(0.02)*	80.28% *(0.01)*	39.03% *(0.02)*
10	48.6% *(0.02)*	48.6% *(0.02)*	82.9% *(0.01)*	47.87% *(0.02)*
15	62.88% *(0.03)*	62.88% *(0.03)*	87.64% *(0.01)*	62.15% *(0.03)*
nothing	71.95% *(0.02)*	71.95% *(0.02)*	90.64% *(0.01)*	70.58% *(0.02)*

Table 25: DecisionTree metrics result for CrossValidation

MaxDepth	Accuracy	Recall	Specificity	F1-Score	CM
3	34.92%	34.92%	78.14%	29.99%	[142 99 0 61; 104 133 0 67; 104 121 0 71; 74 67 0 137]
5	37.29%	37.29%	78.86%	35.23%	[200 45 31 26; 125 69 59 51; 142 60 51 43; 89 26 43 120]
7	42.46%	42.46%	80.56%	40.53%	[212 56 22 12; 122 83 55 44; 124 69 58 45; 62 33 35 148]
10	49.92%	49.92%	83.14%	49.06%	[205 62 20 15; 65 117 80 42; 79 83 88 46; 24 39 36 179]
15	67.2%	67.2%	89.11%	65.99%	[264 19 15 4; 38 146 72 48; 35 62 142 57; 5 12 20 241]
nothing	73.39%	73.39%	91.19%	71.85%	[302 0 0 0; 31 144 84 45; 17 65 164 50; 2 8 12 256]

Table 26: DecisionTree metrics result

Kernel	C	Accuracy	Recall	Specificity	F1-Score
rbf	0.1	30.87% *(0.02)*	30.87% *(0.02)*	77.01% *(0.01)*	27.17% *(0.02)*
rbf	1.0	35.93% *(0.02)*	35.93% *(0.02)*	78.71% *(0.01)*	35.01% *(0.02)*
rbf	10.0	42.2% *(0.01)*	42.2% *(0.01)*	80.79% *(0.0)*	41.5% *(0.01)*
poly	0.1	32.12% *(0.02)*	32.12% *(0.02)*	77.42% *(0.01)*	27.33% *(0.02)*
poly	1.0	35.42% *(0.02)*	35.42% *(0.02)*	78.6% *(0.0)*	33.2% *(0.02)*
linear	0.1	32.5% *(0.02)*	32.5% *(0.02)*	77.56% *(0.01)*	31.06% *(0.02)*
linear	1.0	33.2% *(0.01)*	33.2% *(0.01)*	77.8% *(0.0)*	31.9% *(0.01)*
linear	10.0	32.86% *(0.02)*	32.86% *(0.02)*	77.69% *(0.01)*	31.61% *(0.02)*

Table 27: SVM metrics result for CrossValidation

Kernel	C	Accuracy	Recall	Specificity	F1-Score	CM
rbf	0.1	29.07%	29.07%	76.08%	25.62%	[91 140 7 64; 56 165 15 68; 66 163 10 57; 68 127 6 77]
rbf	1.0	34.66%	34.66%	77.94%	33.64%	[151 84 42 25; 75 125 55 49; 95 100 50 51; 66 88 41 83]
rbf	10.0	40.93%	40.93%	80.06%	40.44%	[171 67 50 14; 71 131 61 41; 74 105 69 48; 46 71 49 112]
poly	0.1	29.49%	29.49%	76.22%	24.41%	[39 184 8 71; 8 211 20 65; 15 209 13 59; 15 172 6 85]
poly	1.0	35.08%	35.08%	77.86%	33.23%	[81 162 43 16; 25 209 40 30; 30 191 45 30; 20 147 32 79]
linear	0.1	30.68%	30.68%	76.63%	28.68%	[111 115 24 52; 63 152 29 60; 86 129 28 53; 63 111 33 71]
linear	1.0	30.85%	30.85%	76.68%	29.24%	[114 104 41 43; 61 147 37 59; 81 131 32 52; 61 105 41 71]
linear	10.0	30.93%	30.93%	76.65%	29.2%	[117 108 43 34; 59 151 43 51; 84 133 31 48; 62 110 40 66]

Table 28: SVM metrics result

Ensemble Model	Parameters	Accuracy	Recall	Specificity	F1-Score
Hard Voting		72.63% *(0.02)*	72.63% *(0.02)*	90.89% *(0.01)*	71.53% *(0.02)*
Stacking (Decision Tree)	Max Depth = None	65.11% *(0.03)*	65.11% *(0.03)*	88.38% *(0.01)*	65.13% *(0.03)*
Stacking (kNN)	numNeighboors = 1	65.32% *(0.03)*	65.32% *(0.03)*	88.45% *(0.01)*	65.25% *(0.03)*
Stacking (SVM)	C = 10, kernel = rbf	74.56% *(0.02)*	74.56% *(0.02)*	91.55% *(0.01)*	74.47% *(0.02)*

Table 29: Ensemble Model metrics result for CrossValidation

Ensemble Model	Final Estimator	Accuracy	Recall	Specificity	F1-Score	CM
Hard Voting		74.24%	74.24%	91.38%	73.11%	[302 0 0 0; 23 187 59 35; 13 106 139 38; 0 16 14 248]
Stacking (Decision Tree)	Max Depth = None	68.98%	68.98%	89.6%	69.16%	[284 18 0 0; 8 162 102 32; 2 99 160 35; 0 34 36 208]
Stacking (kNN)	numNeighboors = 1	69.49%	69.49%	89.78%	69.48%	[288 12 2 0; 8 156 112 28; 2 98 157 39; 0 38 21 219]
Stacking (SVM)	C = 10, kernel = rbf	77.37%	77.37%	92.38%	77.22%	[302 0 0 0; 7 198 83 16; 2 104 171 19; 0 6 30 242]

Table 30: Ensemble Model metrics result

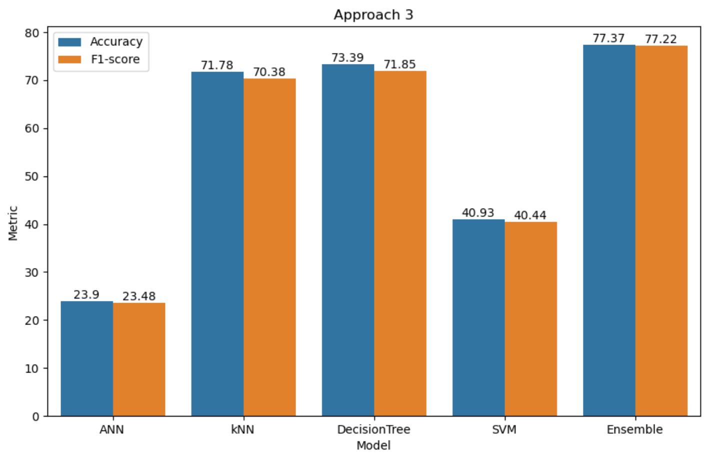

Figure 7: Performance Evaluation: Contrasting Accuracy and F1-Score among Various Models

Artificial Neural Networks The results of the Artificial Neural Networks (ANNs) in the multi-class classification scenario indicate a challenging performance. Across different configurations, including variations in the number of neurons and layers, the ANNs struggled to achieve significant accuracy. Even with changes in architecture, such as changing the number of neurons in layers or combining multiple layers, the ANN models consistently showed limited performance.

During cross-validation (table 21), the ANNs showed modest accuracy scores of around 30% to 33%, reflecting their difficulty in discriminating between multiple classes (lightly better than a random model with 25% accuracy). Furthermore, the final model (table 22) scores reflected these challenges, with accuracy rates ranging from 20% to 25%, highlighting the struggle to effectively classify instances into the various malware classes and benign categories.

The confusion matrices reveal the difficulty of the ANN models in correctly classifying instances into different categories. There's a notable spread of misclassifications, as evidenced by the non-diagonal dominance of the confusion matrices, indicating significant confusion between different classes.

Overall, the ANN models faced significant hurdles in accurately distinguishing and categorising the various malware classes alongside benign instances. Their underwhelming performance underscores the complexity of multi-class classification in this context.

k-Nearest Neighbors The performance of the K-Nearest Neighbours (KNN) models in the multi-class scenario shows different accuracy rates depending on the number of neighbours considered during classification. As the number of neighbours k increases, the accuracy of the models tends to decrease, reflecting the increasing complexity of distinguishing between multiple classes.

Starting with $k = 1$ in the results table 24, the model achieves the highest accuracy at 71.78% , indicating that considering only the nearest neighbour for classification leads to the most accurate predictions within this framework. However, as k increases, the accuracy gradually decreases. This may be indicative of the inherent difficulty in determining class boundaries in a high-dimensional space, as the influence of more neighbours leads to greater class overlap, negatively impacting accuracy.

The recall metric follows the accuracy trend, showing a similar pattern across different k values. Specificity, on the other hand, remains relatively higher, indicating the ability of the models to correctly identify true negatives. However, the F1 score shows a consistent decrease as k increases, indicating a decrease in the models' precision and ability to correctly classify all classes.

The confusion matrices for each k value provide a visual representation of the performance of the models in classifying instances into different categories. The distribution of misclassifications varies, suggesting that confusion between classes increases as k increases.

Overall, the kNN model face challenges in effectively discriminating between multiple classes, especially at higher k values, indicating the limitations of the kNN approach in this multi-class classification task.

Decision Tree The decision tree models, varying in maximum depth, showcased distinct performance trends in the multiclass scenario. In the final results (table 26), commencing with a maximum depth of 3, the models struggled to effectively discern between multiple classes, reflecting in lower accuracy, recall, specificity, and F1-score. However, as the maximum depth increased to 5, 7, 10, 15 and unlimited, there was a noticeable and consistent improvement in model performance across all metrics, highlighting the capacity of deeper trees to better differentiate between classes.

The cross-validation results (25) aligned with the trends observed in the final model evaluations. As the maximum depth increased during cross-validation, there was a concurrent enhancement in all metrics, demonstrating the positive impact of increased tree complexity in handling the multiclass problem effectively. The higher depths consistently led to improved accuracy, recall, specificity, and F1-score, indicating better class separation and predictive capability within the dataset.

The tree with no depth limit exhibited the highest performance among all depths. This scenario underscored the significance of allowing the tree to grow deeper, capturing intricate patterns within the data and resulting in superior classification metrics. This trend further emphasized the importance of an adequately complex decision tree in accurately categorizing instances across multiple classes.

Support Vector Machine The SVM models with different kernels and regularisation parameter C showed variable performance in different configurations. For the cross-validation phase, models with 'rbf' kernel with different C values showed inconsistent results in terms of accuracy, recall, specificity and F1 score. Lower C values and 'poly' and 'linerar' kernels variants consistently produced weaker classification performance.

In the final evaluations (table 28), the 'rbf' kernel with C values of 0.1, 1.0 and 10.0 showed incremental improvements in accuracy, recall, specificity and F1-score. However, despite these improvements, the models struggled to effectively classify instances across multiple classes, as evidenced by the confusion matrices.

The 'poly' kernel, especially at C = 1.0, showed slightly better performance compared to other configurations in the final evaluation. However, its overall performance remained suboptimal in accurately categorising multiclass instances, as shown by the confusion matrices, getting at most 40.93% of accuracy at `C = 10`.

The 'linear' kernel models consistently showed comparable performance metrics across different C values, with no significant improvements. Despite changes in the regularisation parameter, these models struggled to achieve significant improvements in accuracy, as indicated by the confusion matrices.

Essentially, for all SVM configurations investigated, achieving satisfactory classification results in a multi-class scenario was a significant challenge, especially when compared to the performance of other models in the given context. Adjusting the kernels or the regularisation parameter had minimal impact on the overall effectiveness of these SVMs.

In comparison to Decision Trees and k-Nearest Neighbors (kNN), the Support Vector Machines (SVMs) showcased limitations in effectively handling the multiclass classification task. Despite various kernel and regularization parameter configurations, SVMs struggled to achieve competitive performance, falling short in accuracy and overall classification ability when contrasted with the more effective Decision Trees and kNN models.

Ensembles The base models (and hyperparameters) forming the ensembles are Decision Tree ($MaxDepth = None$), kNN ($k = 1$) and SVM ($kernel = rbf, C = 10$).

The ensemble models showed varied performance, each with its own strengths and limitations in different configurations. Hard Voting stood out with a solid 74.24% accuracy, demonstrating reliability in overall predictions. Its high specificity of 91.38% indicates a commendable ability to accurately detect true negatives, which is crucial for accurate benign classifications. However, despite its high accuracy and specificity, the F1 score of 73.11% suggests potential challenges in maintaining a balanced trade-off between precision and recall.

Turning to the stacking models, the decision tree variant achieved a respectable accuracy of 68.98%. This model showed versatility in its predictive capabilities. However, the F1 score of 69.16% may indicate difficulties in capturing nuanced patterns across different classes.

Similarly, the Stacking model using kNN delivered results in line with the Decision Tree model, showing consistency in its predictive performance with an accuracy of 69.49%. However, similar to the Decision Tree based approach, the F1 score of 69.48% suggests potential limitations in effectively capturing diverse data patterns.

In contrast, the Stacking model using SVMs emerged as the standout performer with an accuracy of 77.37% and a robust F1 score of 77.22%. These values represent not only accurate overall predictions, but also an impressive balance between precision and recall, which is crucial when dealing with complex classification scenarios.

The final ensemble results confirmed the superiority of the stacking approach with SVMs. With an accuracy of 77.37% and a strong F1 score of 77.22%, this model significantly outperformed others, excelling in both precision and recall. The Confusion Matrix (CM) reflects its superior performance across different classes, classifying perfectly one malware type and almost the benign class, confirming its potential for real-world applications where accurate identification of multiple classes is critical.

Final result of the approach The conclusive findings from this approach consistently underscore the pivotal role of balanced datasets in achieving reliable and accurate machine learning models, especially in complex, multi-class scenarios. The importance of dataset balancing techniques remains paramount, demonstrating their transformative impact on model performance and highlighting the need for similar datasets.

In addition, ensemble methods consistently emerge as robust solutions, rivalling and often outperforming single base models such as kNN and decision trees. While the inherent complexity of multi-class problems may result in slightly lower accuracy compared to binary scenarios, the ensemble models still show commendable performance. In particular, some models show promising accuracies, leaving room for further refinement and improvement.

2.4 Approach 4: Multiclass Classification with data dimensionality reduction

Based on the results observed in the previous analyses, the next step involves a focused exploration of feature selection and data refinement techniques. Recognising the importance of optimising the dataset to improve model performance, the aim is to identify and incorporate the most influential features, considering methods such as feature selection or data reduction using techniques such as Principal Component Analysis (PCA). This deliberate strategy aims to streamline the dataset by selecting relevant features or applying dimensionality reduction methods, thereby refining the data for improved model training and performance.

Additionally, Principal Component Analysis (PCA) was employed as a complementary technique to further streamline the dataset. PCA serves the purpose of reducing the dimensionality of the data while preserving its key characteristics. By transforming the original features into a new set of uncorrelated variables (principal components), PCA facilitates the identification of patterns and structures within the data. The primary goal here is to condense the information into a smaller number of principal components that capture the maximum variance present in the dataset. This reduction in dimensions not only simplifies the dataset but also aims to mitigate potential noise or redundancy, enhancing the efficiency of subsequent model training and interpretation while maintaining the essential information from the original dataset.

Label	Entries
Benign	1,475
Android_Scareware	1,475
Android_SMS_Malware	1,475
Android_Adware	1,475
Total size	5,900

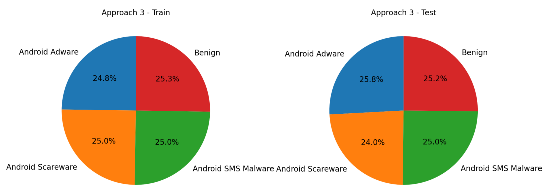

Figure 8: Label distribution fourth approach

We've used the dataset from the previous approach, with a balanced distribution across classes. Table 8 showcases the revised class distribution within this refined dataset. This balanced representation, carefully curated to equalize the number of instances across various classes, aims to mitigate the challenges posed by class imbalances encountered in previous analyses.

2.4.1 Data preprocessing

In this fourth approach, the data processing strategy mirrors the methodologies applied in preceding approaches, upholding consistency in preprocessing techniques. Once again, the data underwent standardization, continuing this practice due to its previously observed effectiveness. This choice stems from prior evidence suggesting that standardization has yielded favorable outcomes in enhancing model performance and convergence. The persistence of this preprocessing step across multiple approaches underscores its reliability and positive impact on the model's learning process, reaffirming its continued utilization in this latest phase.

In addition, the fourth approach retained the strategy of class balance implemented in the second approach. This sustained emphasis on balanced class representation was deemed valuable, especially in the context of a multiclass problem.

2.4.2 Other data related to the experiment

Feature reduction with RFC To implement this strategy of dimensionality reduction, a Random Forest classifier was used to identify the most influential features within the dataset. Using a predetermined feature importance threshold

(0.01 in this case), we identified and extracted these significant features. This refined selection was then used to curate a new, condensed dataset containing only these salient features. This deliberate curation aimed to construct a streamlined dataset that encapsulated the essence of the original data, while discarding redundant or less influential features. This refined dataset, synthesised by merging the most discriminative features, is intended to serve as an optimised input for subsequent model training and evaluation phases.

Once the Random Forest Classifier was trained using the train dataset, a feature importance analysis was conducted. Features with an importance score exceeding 0.01 were selected, resulting in a subset of 38 significant features. This rigorous filtration process aimed to distill the most influential attributes, streamlining the dataset to include only those elements deemed crucial by the classifier. The utilization of these 38 selected features ensures a more focused and refined dataset, pretending to optimize the model's learning process by concentrating solely on the most discriminative attributes.

PCA Implementation The Principal Component Analysis (PCA) was conducted with a specified number of components set to 2. Initially fitted solely with the training set, the PCA was then applied to transform both the training and test input datasets. This process of dimensionality reduction, narrowing down the features to just two principal components, was performed consistently on both the training and testing data. The objective behind this step was to condense the information present in the datasets while retaining essential patterns and structures, allowing for a more efficient representation of the data in a reduced space.

Cross Validation Continuity in the data handling process persisted into the fourth approach, aligning with the methodologies employed in earlier stages. Robust model evaluation endured with the consistent use of a 10-fold cross-validation technique, guaranteeing extensive validation across varied dataset subsets. This approach remained dedicated to comprehensive validation, facilitating a thorough evaluation of model performance and its ability to generalize across distinct sections of the dataset.

Train-test split A 20% data split was reserved for testing purposes using the hold-out method, maintaining consistency in the assessment approach.

Feature Engineering Similar to prior iterations, the fourth approach followed analogous feature selection and engineering methodologies. This involved systematic removal of columns with minimal information and an IP address transformation to enhance data interpretability. The resulting dataset comprised 70 refined columns essential for subsequent model training and evaluation before the reduction with RFC were done.

ANN hyperparameters Consistency persisted in preserving the hyperparameters for the Artificial Neural Network (ANN) model in the fourth approach. These settings remained unchanged, retaining the same configuration established in the initial approach. This deliberate choice aimed to establish a consistent and standardized modeling framework across all experiments, enabling direct comparison of model performances and outcomes across different approaches.

Model selection for ensembles criteria In line with previous practices detailed in the third approach, the fourth iteration also prioritized accuracy and F1 score metrics for ensemble model selection. This ongoing emphasis aimed to identify robust models proficient in multiclass accuracy while balancing precision and recall across individual classes, as previously highlighted.

2.4.3 Results

During this section, we will be displaying a series of tables that showcase various outcomes.

Architecture	Accuracy	Recall	Specificity	F1-Score
20	29.31% *(0.01)*	29.31% *(0.01)*	76.42% *(0.0)*	26.94% *(0.01)*
40	28.93% *(0.01)*	28.93% *(0.01)*	76.29% *(0.0)*	25.32% *(0.01)*
80	28.04% *(0.01)*	28.04% *(0.01)*	76.0% *(0.0)*	22.05% *(0.01)*
100	27.94% *(0.01)*	27.94% *(0.01)*	75.97% *(0.0)*	21.27% *(0.01)*
60, 120	27.55% *(0.01)*	27.55% *(0.01)*	75.83% *(0.0)*	17.43% *(0.01)*
80, 50	28.04% *(0.01)*	28.04% *(0.01)*	76.0% *(0.0)*	19.56% *(0.01)*
80, 100	27.5% *(0.01)*	27.5% *(0.01)*	75.85% *(0.0)*	17.23% *(0.01)*
100, 40	27.92% *(0.01)*	27.92% *(0.01)*	75.98% *(0.0)*	19.62% *(0.01)*

Table 31: ANN metrics result for CrossValidation

Architecture	Accuracy	Recall	Specificity	F1-Score	CM
20	24.92%	24.92%	74.6%	19.58%	[119 38 129 16; 117 11 165 11; 101 23 156 16; 123 27 120 8]
40	26.36%	26.36%	75.06%	19.43%	[144 12 136 10; 129 2 164 9; 131 4 155 6; 145 5 118 10]
80	20.59%	20.59%	72.66%	15.58%	[98 187 10 7; 152 133 12 7; 143 139 5 9; 152 113 6 7]
100	24.24%	24.24%	73.94%	18.08%	[123 158 9 12; 137 152 7 8; 143 138 5 10; 143 120 9 6]
60, 120	23.56%	23.56%	76.42%	9.45%	[0 0 5 297; 0 0 1 303; 0 0 3 293; 0 0 3 275]
80, 50	22.63%	22.63%	75.09%	14.61%	[0 120 0 182; 0 96 0 208; 0 95 0 201; 0 106 1 171]
80, 100	22.12%	22.12%	75.32%	15.08%	[26 20 25 231; 46 2 30 226; 39 7 29 221; 42 3 29 204]
100, 40	25.0%	25.0%	75.84%	18.54%	[12 20 130 140; 15 4 129 156; 13 3 125 155; 15 4 105 154]

Table 32: ANN metrics result

k	Accuracy	Recall	Specificity	F1-Score
1	70.96% *(0.02)*	70.96% *(0.02)*	90.3% *(0.01)*	69.58% *(0.02)*
2	60.53% *(0.02)*	60.53% *(0.02)*	86.85% *(0.01)*	58.97% *(0.02)*
3	55.3% *(0.02)*	55.3% *(0.02)*	85.1% *(0.01)*	53.46% *(0.02)*
5	50.93% *(0.01)*	50.93% *(0.01)*	83.65% *(0.0)*	48.64% *(0.02)*
7	46.38% *(0.01)*	46.38% *(0.01)*	82.14% *(0.0)*	44.3% *(0.02)*
10	41.87% *(0.02)*	41.87% *(0.02)*	80.63% *(0.01)*	40.58% *(0.02)*
15	39.11% *(0.03)*	39.11% *(0.03)*	79.71% *(0.01)*	38.46% *(0.03)*

Table 33: kNN metrics result for CrossValidation

k	Accuracy	Recall	Specificity	F1-Score	CM
1	70.34%	70.34%	90.17%	68.79%	[302 0 0 0; 25 130 89 60; 13 83 156 44; 6 12 18 242]
2	61.36%	61.36%	87.08%	59.74%	[297 5 0 0; 25 169 50 60; 13 134 81 68; 6 85 10 177]
3	54.58%	54.58%	84.87%	52.79%	[297 5 0 0; 39 110 77 78; 36 92 98 70; 15 80 44 139]
5	50.68%	50.68%	83.61%	48.47%	[281 5 12 4; 53 104 73 74; 49 78 80 89; 28 68 49 133]
7	47.54%	47.54%	82.49%	45.1%	[264 11 20 7; 75 92 66 71; 73 68 78 77; 44 58 49 127]
10	40.42%	40.42%	80.11%	38.67%	[208 39 27 28; 88 86 58 72; 78 71 64 83; 62 58 39 119]
15	35.42%	35.42%	78.47%	34.56%	[157 52 52 41; 86 81 64 73; 77 76 61 82; 51 55 53 119]

Table 34: kNN metrics result

MaxDepth	Accuracy	Recall	Specificity	F1-Score
3	29.54% *(0.02)*	29.54% *(0.02)*	76.47% *(0.01)*	25.2% *(0.02)*
5	33.03% *(0.02)*	33.03% *(0.02)*	77.66% *(0.01)*	29.97% *(0.03)*
7	36.16% *(0.02)*	36.16% *(0.02)*	78.74% *(0.01)*	34.58% *(0.02)*
10	45.68% *(0.02)*	45.68% *(0.02)*	81.89% *(0.01)*	44.73% *(0.02)*
15	59.39% *(0.03)*	59.39% *(0.03)*	86.45% *(0.01)*	58.37% *(0.03)*
nothing	70.55% *(0.03)*	70.55% *(0.03)*	90.17% *(0.01)*	69.12% *(0.03)*

Table 35: DecisionTree metrics result for CrossValidation

MaxDepth	Accuracy	Recall	Specificity	F1-Score	CM
3	28.22%	28.22%	76.38%	24.56%	[64 106 4 128; 36 137 4 127; 24 139 7 126; 34 115 4 125]
5	32.37%	32.37%	77.96%	30.92%	[112 37 33 120; 50 83 38 133; 36 85 34 141; 35 62 28 153]
7	35.59%	35.59%	78.81%	34.47%	[160 28 35 79; 73 60 71 100; 64 54 72 106; 50 39 61 128]
10	45.85%	45.85%	81.89%	44.12%	[233 32 6 31; 76 107 58 63; 62 90 74 70; 52 68 31 127]
15	60.68%	60.68%	86.92%	59.9%	[265 17 10 10; 29 125 87 63; 24 81 136 55; 14 44 30 190]
nothing	70.76%	70.76%	90.33%	69.26%	[302 0 0 0; 25 134 88 57; 17 67 161 51; 8 16 16 238]

Table 36: DecisionTree metrics result

Kernel	C	Accuracy	Recall	Specificity	F1-Score
rbf	0.1	27.65% *(0.02)*	27.65% *(0.02)*	75.97% *(0.01)*	21.56% *(0.02)*
rbf	1.0	28.73% *(0.02)*	28.73% *(0.02)*	76.31% *(0.01)*	24.59% *(0.02)*
rbf	10.0	30.15% *(0.02)*	30.15% *(0.02)*	76.75% *(0.01)*	27.09% *(0.02)*
poly	0.1	26.19% *(0.01)*	26.19% *(0.01)*	74.94% *(0.0)*	13.41% *(0.01)*
poly	1.0	26.23% *(0.01)*	26.23% *(0.01)*	74.96% *(0.0)*	13.51% *(0.01)*
linear	0.1	26.59% *(0.02)*	26.59% *(0.02)*	75.63% *(0.01)*	17.33% *(0.02)*
linear	1.0	26.59% *(0.02)*	26.59% *(0.02)*	75.62% *(0.01)*	17.32% *(0.02)*
linear	10.0	26.59% *(0.02)*	26.59% *(0.02)*	75.62% *(0.01)*	17.32% *(0.02)*

Table 37: SVM metrics result for CrossValidation

Kernel	C	Accuracy	Recall	Specificity	F1-Score	CM
rbf	0.1	28.39%	28.39%	75.76%	22.23%	[51 189 0 62; 23 217 2 62; 23 217 1 55; 42 170 0 66]
rbf	1.0	26.86%	26.86%	75.37%	23.77%	[37 161 38 66; 15 183 42 64; 13 192 34 57; 25 151 39 63]
rbf	10.0	29.66%	29.66%	76.46%	27.31%	[67 132 21 82; 35 167 17 85; 37 161 28 70; 41 127 22 88]
poly	0.1	24.07%	24.07%	76.47%	12.1%	[16 6 0 280; 5 4 2 293; 7 5 1 283; 7 6 2 263]
poly	1.0	24.07%	24.07%	76.47%	12.11%	[16 6 0 280; 5 4 2 293; 7 5 1 283; 6 7 2 263]
linear	0.1	26.61%	26.61%	75.22%	16.94%	[0 220 0 82; 0 234 1 69; 0 235 2 59; 0 200 0 78]
linear	1.0	26.69%	26.69%	75.24%	17.13%	[0 220 0 82; 0 234 2 68; 0 236 3 57; 0 200 0 78]
linear	10.0	26.69%	26.69%	75.24%	17.13%	[0 220 0 82; 0 234 2 68; 0 236 3 57; 0 200 0 78]

Table 38: SVM metrics result

Ensemble Model	Parameters	Accuracy	Recall	Specificity	F1-Score
Hard Voting		73.41% *(0.02)*	73.41% *(0.02)*	91.15% *(0.01)*	72.32% *(0.03)*
Stacking (Decision Tree)	Max Depth = None	65.49% *(0.01)*	65.49% *(0.01)*	88.5% *(0.0)*	65.46% *(0.01)*
Stacking (kNN)	numNeighboors = 1	66.38% *(0.02)*	66.38% *(0.02)*	88.8% *(0.01)*	66.3% *(0.02)*
Stacking (SVM)	C = 10, kernel = rbf	74.92% *(0.02)*	74.92% *(0.02)*	91.67% *(0.01)*	74.81% *(0.02)*

Table 39: Ensemble Model metrics result for CrossValidation

Ensemble Model	Final Estimator	Accuracy	Recall	Specificity	F1-Score	CM
Hard Voting		74.41%	74.41%	91.43%	73.22%	[302 0 0 0; 23 192 55 34; 13 108 136 39; 2 20 8 248]
Stacking (Decision Tree)	Max Depth = None	68.14%	68.14%	89.4%	67.93%	[288 8 4 2; 7 153 90 54; 1 104 148 43; 0 35 28 215]
Stacking (kNN)	numNeighboors = 1	66.86%	66.86%	88.92%	66.74%	[290 8 4 0; 8 152 104 40; 1 116 136 43; 0 35 32 211]
Stacking (SVM)	C = 10, kernel = rbf	74.83%	74.83%	91.47%	74.6%	[302 0 0 0; 7 194 89 14; 1 135 143 17; 0 22 12 244]

Table 40: Ensemble Model metrics result

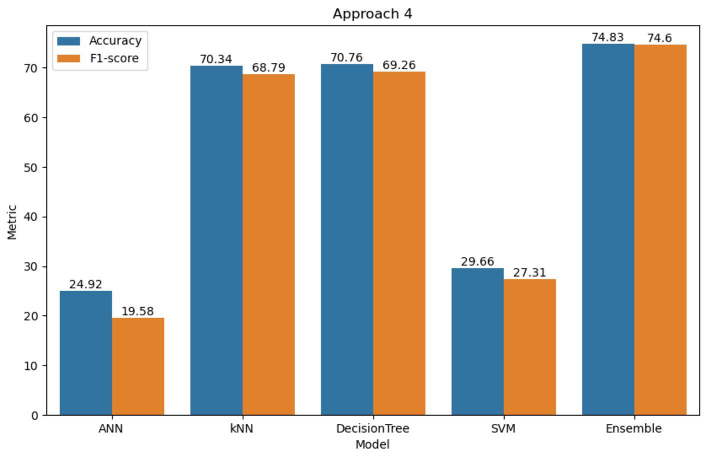

Figure 9: Performance Evaluation: Contrasting Accuracy and F1-Score among Various Models

Artificial Neural Networks Table 31 displays the performance metrics of an Artificial Neural Network (ANN) model assessed through cross-validation with varying architectural configurations. The ANN models were trained and evaluated on the dataset using different architecture settings. The results indicate that the model achieved moderate Accuracy ranging from 27.5% to 29.31% for all architectures tested. The Recall values were consistent across all architectures at approximately 27.5% to 29.31%. Specificity remained stable around 75% to 76.42%, while the F1-Score ranged from 17.23% to 26.94%. These findings suggest that the model had a moderate performance across different architectural configurations.

Table 32 displays some results indicating that different architecture combinations produce varied performance. Accuracy values range from 20.59% to 25.0%, while Recall values show a similar trend at around 20.59% to 25.0%. Specificity remains between 72.66% to 76.42%, and the F1-Score varies between 9.45% to 18.54%. The Confusion Matrix demonstrates varied distributions across different classes for each configuration, highlighting the model's struggle in distinguishing between classes.

To sum up, after analyzing various configurations of ANN architecture, it has been identified that there are areas that require improvement. This indicates the need for further research into architectural enhancements and possible model modifications. Tackling these challenges could lead to the development of more precise and dependable malware detection systems, especially for datasets with imbalanced class distributions.

k-Nearest Neighbors Table 33 demonstrates the performance metrics of the kNN model evaluated through cross-validation for different values of k. As k increases, the accuracy, recall, specificity, and F1-score metrics gradually decrease, indicating a degradation in model performance as the number of neighbors considered for classification grows.

The following table (Table 34 displays the performance metrics of the kNN model for different values of k. Generally, as the value of k increases, the accuracy, recall, specificity, and F1-score tend to decrease. Additionally, the confusion matrices (CM) for each k value show different distributions across various classes, which indicates that the model faces difficulty in distinguishing between classes as the number of considered neighbours increases.

Based on the findings, it can be concluded that the kNN model performs satisfactorily with lower values of k. However, as the value of k increases, the model's performance starts to decline. This decline can be attributed to the model's increased dependence on a larger number of neighbors, which ultimately leads to a reduction in the model's ability to generalize well to new data.

Decision Tree Table 35 shows the performance metrics of a Decision Tree model evaluated through cross-validation. The model has varying maximum depths, and as the depth increases, the accuracy, recall, specificity, and F1-score also increase. The model with no maximum depth constraint, which means unrestricted tree growth, has the highest performance across all metrics, achieving an accuracy of 70.55% and an F1-score of 69.12%.

Table 36 demonstrates the Decision Tree model's performance for different maximum depths. The model shows improved performance as the maximum depth increases, with higher accuracy, recall, specificity, and F1-score. The confusion matrices (CM) reveal different distributions across classes, highlighting the model's ability to differentiate between classes more effectively with increased depth. The model without a maximum depth constraint shows superior performance, achieving an accuracy of 70.76% and an F1-score of 69.26%.

The decision tree without a depth restriction (`MaxDepth`) has shown the most promising results in terms of accuracy and F1-score when compared to the trees with depth limitations. This is possible because the depth-unrestricted trees are more complex and can learn more intricate patterns in the training data.

Support Vector Machine Table 37 displays the cross-validation results for different kernel and C parameter configurations in the Support Vector Machine (SVM) algorithm. Overall, there is no substantial improvement in performance metrics when varying the kernel or the value of C. The metrics, such as accuracy, recall, specificity, and F1-Score, vary within a narrow range across different configurations.

Additionally, Table 38 presents SVM results with different kernel and C settings without cross-validation. A similar trend is observed here, where the performance metrics show similar values across various kernel and C combinations.

It's worth noting that the values of accuracy, recall, specificity, and F1-Score in both tables remain close to each other regardless of the kernel and C configurations. This indicates that these parameters do not significantly influence the SVM model's performance on this dataset. Similarly, regardless of the configurations selected for the kernel and parameter C, the SVM model fails to achieve a good capability to correctly classify all classes within the dataset.

Ensembles In Table 39, we have displayed the metrics of the cross-validation ensemble models. As per the table, the Hard Voting ensemble model exhibited remarkable F1-score of around 72.32%, with a specificity of 91.15%. While the Stacking models with Decision Tree and kNN demonstrated moderate performance with F1-score hovering around 65-66% and varying specificities, the Stacking with SVM showcased the highest accuracy (74.92%) and F1-score (74.81%), indicating its superior performance among the cross-validation ensemble models.

Table 40 shows the ensemble model metrics. The trends observed in this table are consistent, with the Hard Voting ensemble model sustaining a solid performance (accuracy 74.41%, F1-score

73.22%), followed by the SVM-based Stacking model (accuracy 74.83%, F1-score 74.6%). However, the Stacking models with Decision Tree and kNN displayed less impressive results compared to the SVM and Hard Voting ensembles, exhibiting accuracies and F1-score around 66-68%. To summarize, SVM-based ensembles and Hard Voting emerged as the most effective models, displaying superior accuracy and F1-scores, while Stacking models with Decision Tree and kNN showcased comparatively moderate performance, yet falling short of the SVM and Hard Voting ensembles in terms of accurately classifying the dataset samples.

Final result of the approach During the comprehensive evaluation of various machine learning models for malware detection, it was observed that Artificial Neural Networks (ANNs) delivered moderate performance with different architectural configurations. Although these models showed consistent but moderate accuracy and recall, they had difficulties in producing robust results with F1-Scores ranging from 17% to 29%. Similarly, the k-Nearest Neighbors (kNN) model showcased a decrease in performance as the number of neighbors considered increased. This decline in accuracy, recall, specificity, and F1-Score highlighted the model's obstacle in distinguishing between classes with larger k values.

On the other hand, Decision Trees demonstrated significant results, exhibiting higher accuracy and F1-Scores as the maximum depth increased. The Decision Tree without a depth constraint demonstrated a promising accuracy of 70.76% and an F1-Score of 69.26%, indicating its effectiveness in detecting complex patterns in the dataset. Conversely, the Support Vector Machine (SVM) model, despite varying kernel and C parameters, displayed consistent performance across metrics, indicating that the parameters had limited influence on classification outcomes.

Regarding the ensemble models, the Hard Voting ensemble and SVM-based Stacking models stood out with superior accuracy and F1-Scores. Hard Voting consistently showed an F1-Score of around 73%, while the SVM-based Stacking model achieved an F1-Score of approximately 74%. However, Stacking models with Decision Trees and kNN presented moderate results, lagging behind the SVM and Hard Voting ensembles in accurately classifying dataset samples.

3 Discussion and future work

Our study presents an Android malware detection system using machine learning methods. Through rigorous testing, our system has demonstrated a commendably low rate of false negatives, demonstrating the effectiveness of a balanced data approach. However, a notable area for improvement lies in mitigating the prevalence of false positives, which remains relatively high.

The inherent imbalance in our dataset, with only 7% representing the benign class, suggests the prospect of exploring anomaly detection methods as a promising avenue for future research. Addressing this imbalance could potentially lead to improved precision and accuracy in the detection of benign cases. This recommendation opens doors for further exploration and refinement, with the aim of strengthening the reliability and overall performance of the system.

A pivotal insight derived from our study emphasizes the indispensable role of class balance, vividly showcased in the divergent metrics between approach 1 and approach 2. This striking contrast underscores the pivotal significance of a balanced dataset in cultivating more accurate and dependable models for Android malware detection. Notably, when juxtaposing both approaches, we discern a spectrum in model performance, ranging from models exhibiting nearly 100-0 recall and specificity to others showcasing over 90% in each metric. Among these, a standout ensemble model achieved 100% specificity and an impressive 98.04% recall, marking a resounding triumph in the binary identification of malware.

In tackling the challenge of multi-class classification, our focus shifted to identifying specific types of malware, yielding discrete metrics of 77% for accuracy and F1 score in an ensemble model. Furthermore, in a dataset reduced by PCA and feature selection, the metrics hovered around 74% for accuracy and F1 score. These results highlight the potential for improvement and indicate that there is room for improvement in model performance. Nevertheless, it's crucial to contextualise these results within the initial state of the dataset, acknowledging its inherent complexity and the relatively limited representation of certain classes. Given the inherent challenges posed by the composition of the dataset, the metrics achieved demonstrate a notable foundation for further progress in identifying different types of malware.

Ensemble models emerged as the central cornerstone of our approach, with the stacking technique in particular emerging as the best performing model. This ensemble framework, encapsulating different base models, provided compelling metrics that demonstrated the significant advances that could be achieved through model fusion. The ability of the stacking model to utilise different base models enabled the achievement of commendable metrics, illustrating the ability of ensemble techniques to combine different strengths and produce more robust and reliable predictions. This reaffirms the importance and remarkable effectiveness of ensemble methods in harnessing the collective power of individual models, ultimately leading to significant improvements in performance and predictive capability.

Looking ahead, a compelling avenue for future research is to train on an expanded dataset. Currently, our system operates on a limited portion of the available data, which provides an opportunity to refine the models using a more comprehensive dataset. However, given the time constraints, an exhaustive expansion is currently not feasible.

Furthermore, the inclusion of novel features is a noteworthy consideration. As noted in the existing literature, the rapid obsolescence of machine learning models in software detection within approximately six months necessitates adaptability. A pipeline for continuous data updates and a model that continuously learns could be an interesting prospect, ensuring that the system remains up to date with evolving malware trends.

In addition, an exciting prospect for future work is to explore a wider range of hyperparameters and innovative techniques to further improve the performance metrics of multi-class models. The aim is to push the boundaries and aim for metrics to get as much as posible, aiming for significant improvements in accuracy and f1-score across all classes.

References

[1] Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., and Rieck, K. Drebin: Effective and explainable detection of android malware in your pocket.

[2] by Statista Research Department, P. Mobile os market share worldwide 2009-2023, Oct 2023.

[3] Chen, Y., Ding, Z., and Wagner, D. Continuous learning for android malware detection, 2023.

[4] Chowdhury, M., Rahman, A., and Islam, M. R. Malware analysis and detection using data mining and machine learning classification. pp. 266–274.

[5] Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M. S., Conti, M., and Rajarajan, M. Android security: A survey of issues, malware penetration, and defenses. *IEEE Commun. Surv. Tutorials 17*, 2 (2015), 998–1022.

[6] Feng, R., Lim, J. Q., Chen, S., Lin, S.-W., and Liu, Y. Seqmobile: An efficient sequence-based malware detection system using rnn on mobile devices. *2020 25th International Conference on Engineering of Complex Computer Systems (ICECCS)* (2020), 63–72.

[7] Lashkari, A. H., Kadir, A. F. A., Taheri, L., and Ghorbani, A. A. Toward developing a systematic approach to generate benchmark android malware datasets and classification. In *2018 International Carnahan Conference on Security Technology (ICCST)* (2018), pp. 1–7.

[8] Liu, S., Peng, G., Zeng, H., and Fu, J. A survey on the evolution of fileless attacks and detection techniques. *Computers Security* (2023), 103653.

[9] Wu, B., Chen, S., Gao, C., Fan, L., Liu, Y., Wen, W., and Lyu, M. R. Why an android app is classified as malware: Toward malware classification interpretation. *ACM Trans. Softw. Eng. Methodol. 30*, 2 (mar 2021).