SSRG International Journal of Electronics and Communication Engineering
ISSN: 2348-8549/ https://doi.org/10.14445/23488549/IJECE-V11I9P122

Volume 11 Issue 9, 253-269, September 2024
© 2024 Seventh Sense Research Group®

Original Article

Enhancing Android Malware Detection: A Grid-Tuned Two-Layered Stacking Approach

Ravi Eslavath[1], Upendra Kumar Mummadi[2]

[1]Department of CSE, University College of Engineering, Osmania, University, Hyderabad, India.
[2]Department of CSE, Muffakham Jah College of Engineering and Technology, Hyderabad, India.

[1]Corresponding Author : eslavathravi@gmail.com

Received: 12 July 2024 Revised: 22 August 2024 Accepted: 13 September 2024 Published: 30 September 2024

Abstract *- Android malware detection is critical for protecting users from cybercrime by automatically identifying potentially harmful applications before they can affect devices. This study explores the efficacy of various machine learning techniques, including ensemble and voting algorithms, for enhancing malware detection. Traditional methods face challenges due to the increasing number of attributes and the dynamic nature of certain features, necessitating more robust solutions. The proposed model addresses these challenges by initially transforming class labels into numerical format and applying normalization to independent attributes, thereby reducing variance and improving computational efficiency. The methodology involves a two-layered stacking approach rather than a single-layer model to minimize the risk of misclassification and improve the handling of unknown malware. At the base level, hyperparameters of traditional classifiers such as SVM, KNN, and Bernoulli Naive Bayes are finely tuned using repeated cross-validation, creating a diverse meta data repository. The stacking classifier employs a voting mechanism that considers all possible true and false classification rates, enhancing predictive accuracy. The next layer (meta classifier-1) utilizes tuned ensemble methods to generate numerical predictions, which are then processed by a final logistic regression layer (meta classifier-2). The proposed model demonstrates a significant improvement, achieving a +0.9% increase in accuracy compared to standalone tuning algorithms, thereby offering a more reliable and efficient approach to Android malware detection. This study utilizes the Drebin dataset, which includes 15,036 samples comprising 5,560 malware and 9,476 benign applications, to evaluate the model's performance.*

Keywords *- Bernoulli NB, 2- layered stack, Meta data, Hyperparameters, Malware analysis.*

1. Introduction

In the contemporary digital era, the proliferation of smartphones has revolutionized communication, information access, and entertainment. Among these, Android has emerged as the dominant operating system, powering billions of devices globally. However, this ubiquity has made Android a prime target for malicious software or malware, which poses significant threats to users' privacy, security, and financial well-being. As cybercriminals continually develop new and sophisticated malware variants, the challenge of effective detection and mitigation becomes increasingly complex and critical. Android malware detection is a vital aspect of cybersecurity, aimed at identifying and neutralizing malicious applications before they can inflict harm. Traditional detection methods, such as signature-based and heuristic approaches, often fall short in detecting novel and evolving threats due to their reliance on predefined patterns. Consequently, there has been a paradigm shift towards Machine Learning (ML) techniques [1], which offer the potential to learn and adapt from vast datasets, thereby improving detection rates and reducing false positives.

Machine learning algorithms, particularly those utilizing ensemble and stacking techniques, have shown promise in enhancing malware detection accuracy. Ensemble methods combine multiple classifiers to make more robust predictions while stacking. A specific type of ensemble learning integrates various models at different levels to capitalize on individual strengths and mitigate weaknesses. Despite these advancements, the dynamic nature of malware attributes presents ongoing challenges, necessitating continuous refinement of detection models.

Despite these advancements, the dynamic nature of malware attributes presents ongoing challenges. Existing machine learning algorithms, while promising, still struggle with the high dimensionality and variability of malware data. This study introduces a novel Grid Tuned Two-Layered Stacking Approach for Malware Detection (GTTSAMD). By leveraging grid search optimization and repeated cross-validation, the proposed framework aims to fine-tune hyperparameters and enhance model performance, thereby addressing the limitations of current methods. The primary problem addressed by this study is the inadequate detection accuracy and adaptability of existing Android malware detection models when confronted with new and evolving malware variants. Traditional methods [2] and some machine learning models often result in high false positive rates and fail to generalize well to unseen data. This research seeks to fill this gap by developing a more robust and accurate detection framework. The primary problem addressed by this study is the inadequate detection accuracy and adaptability of existing Android malware detection models when confronted with new and evolving malware variants. Traditional methods and some machine learning models often result in high false positive rates and fail to generalize well to unseen data. This research seeks to fill this gap by developing a more robust and accurate detection framework [3].

1. How can grid search optimization enhance the performance of base-level machine learning classifiers in malware detection?
2. What is the impact of a two-layered stacking approach on the overall detection accuracy and robustness of the model?
3. How does the proposed model compare to traditional single-layer and ensemble models in terms of precision, recall, and false positive rates?

This research is significant as it addresses a critical need in the field of cybersecurity: improving the detection of Android malware. By enhancing detection accuracy and reducing false positives, the proposed model can better protect users from cyber threats. The study's findings could have broad implications for the development of more sophisticated malware detection systems and contribute to safer digital environments for Android users worldwide.

The structure of the paper is as follows: Section 2, background of the study and Section 3, literature survey, reviews previous research on machine learning techniques for android malware detection, highlighting their merits and limitations. Section 4, the proposed methodology, details the two-layered stacking approach, including the selection and tuning of base and meta-level classifiers and the grid search optimization process. Section 5, results and discussion, presents the experimental setup, evaluation metrics, and a comprehensive analysis of the model's performance compared to traditional approaches. Section 6, conclusion, summarizes the key findings, discusses the implications of the research, and suggests directions for future work. By systematically addressing these elements, this paper aims to provide a comprehensive and insightful contribution to the field of Android malware detection.

2. Background

Android malware detection is a crucial task in identifying and mitigating cybercrimes in the digital world. Given the complexity of this problem, researchers have turned to advanced machine learning techniques to enhance detection accuracy. One such approach is the Gaussian Naive Bayes (GNB) algorithm [4], which has been widely utilized for both binary and multiclass classification tasks due to its efficiency and simplicity.

GNB is particularly effective in text classification problems involving large feature spaces, making it suitable for Android malware detection. In this context, GNB can classify an Android application as either malicious or benign based on a variety of features, such as permissions requested by the application, API calls made [5], and specific patterns in the code.

To train a GNB model for Android malware detection, a dataset of Android applications labeled as either malware or benign is required. The model extracts features from each application in the dataset and uses these to train the GNB classifier. Once trained, the model can classify new applications by extracting the same set of features and applying the GNB model to the feature vector, as shown in the following equation:

$$\mathrm{P(Xi|Y='S\_B')} = \frac{1}{\sqrt{2}*\pi*\sigma^2} * e^{\frac{-(\sum X_i - \mu)^2}{2*\sigma^2}} \tag{1}$$

While GNB is a powerful tool for malware detection, its accuracy is highly dependent on the quality of the features and the training dataset. Therefore, ensuring high-quality data is crucial for achieving reliable results.

To address the limitations of traditional approaches, this study proposes a model that utilizes stacking algorithms tuned with the Grid Search approach [6]. Stacking is an ensemble learning technique that combines multiple individual models to improve overall predictive performance. This method leverages the strengths of different models, leading to better generalization and robustness. Stacking also allows for greater flexibility by integrating models with different architectures or trained on various data types, which can reduce variance and provide a more stable estimate of the underlying relationships in the data.

The proposed model addresses these issues using stacking algorithms tuned with Grid Search. Stacking is an ensemble learning technique that combines multiple individual models to improve overall predictive performance. By leveraging the strengths and mitigating the weaknesses of different models, stacking can enhance accuracy, generalization, robustness, and flexibility. This approach allows for the integration of various models with different architectures or training on different data types. Additionally, stacking can reduce variance and provide a more stable estimate of the underlying relationship between input and output variables. It also offers improved interpretability and scalability, making it suitable for large datasets and real-time applications.

The process involved in stacking is illustrated in Figure 1. Base prediction algorithms, known as "Level-0" models, predict elements based on training data, which is then cross-validated to construct meta data. A meta-classifier algorithm is applied to this meta data for evaluation. Some thumb rules for the stacking process include ensuring the meta-classifier evaluates data not trained by any base prediction model, using two-layered stacks due to a large amount of data, and performing cross-validation with the Repeated K-Fold method. This method repeats the K-Fold process several times with different folds [7], providing a robust evaluation and reducing the risk of overfitting by testing the model on various data subsets. The proposed model employs 5-fold cross-validation, as shown in Figure 2.

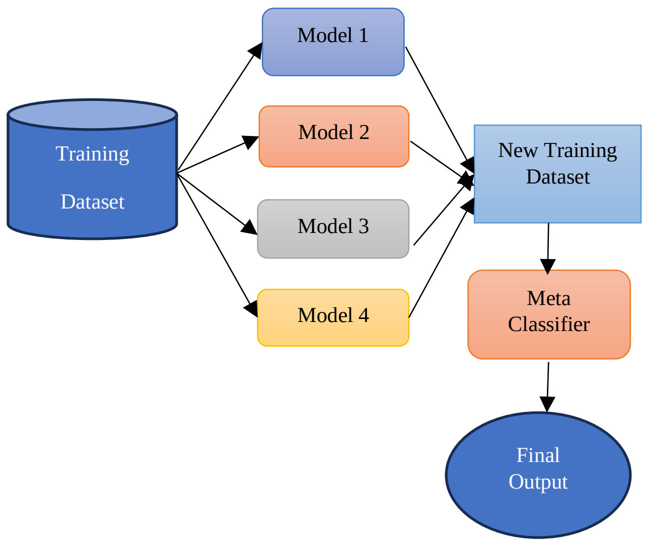

Fig. 1 Stacking algorithm workflow to classify malware

The reliability and validity of any machine learning model deployment are crucial. Reliability is measured using kappa statistics, which evaluate the agreement between different models on the same training data. The kappa score is calculated as shown in Equation (2):

$$KS(Record[i]) = \frac{Number\ of\ models\ suggested\ (Yes) - Number\ of\ models\ probable(Yes)}{1 - Number\ of\ models\ Probable(yes)} \quad (2)$$

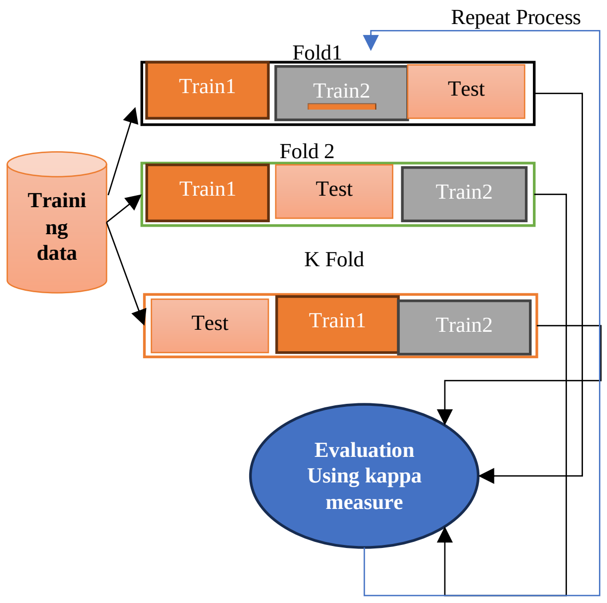

Fig. 2 5-Fold cross validation with kappa evaluation

The process involves several key steps:

- Data Transformation and Normalization: Transform class labels and normalize independent attributes to reduce variability.
- Cross-Validation: Apply repeated K-Fold cross-validation to create robust meta data and evaluate the model's performance.
- Grid Search Optimization: Tune the hyperparameters of the base and meta-classifiers to enhance model accuracy.
- Stacking Ensemble: Combine the models using a voting mechanism to improve overall predictive performance.

The proposed model employs 5-fold cross-validation to ensure a reliable assessment of model performance, mitigating the risk of overfitting and reducing the impact of random fluctuations in the results. The reliability of the model is measured using the kappa statistic, which evaluates the level of agreement between different models on the same training data.

3. Literature Survey

The literature on Android malware detection reveals various innovative approaches leveraging machine learning to enhance detection accuracy and robustness. Researchers have explored diverse techniques, from traditional algorithms like Naive Bayes and Support Vector Machines to advanced methods such as deep learning and ensemble learning. This section provides a concise overview of significant contributions in this field, highlighting the methodologies, key findings, and performance metrics of notable studies. These insights lay the groundwork for understanding the evolution of malware detection strategies and underscore the necessity for continuous improvement in combating sophisticated cyber threats. Suleiman Y. Yerima et al. [8] proposed machine learning methods to analyze malware-affected Android devices, highlighting the transmission of viruses through links, applications, and .apk files. Their approach involved parallel classifications using a matrix of four different techniques: rule-based, function-based, tree-based, and probabilistic methods. The initial data was trained using vectors from new applications, and intermediate outputs were predicted before combining classifications. This ensemble included decision trees, Naive Bayes, and PART (a rule-based method similar to decision trees but utilizing decision lists). Their evaluation, employing three validation techniques, achieved an accuracy of 96.3% in the PART approach.

Rishab Agrawal et al. [9] addressed the challenge of identifying new malware on Android phones. They performed both semantic and permission-based analyses, utilizing a system with separate admin and user portals. The admin panel managed .apk files and comments, while the user panel allowed app uploads for analysis. Six attributes related to .apk files were considered, and both malware and semantic data were analyzed concurrently. This system effectively identified malware-infected devices and generated necessary permissions, achieving notable classification accuracy.

Long Wen et al. [10] identified the limitations of traditional methods in recognizing signatures in unknown applications. They adopted a machine learning approach, combining static and dynamic analyses to extract features. Principal Component Analysis (PCA) was used for feature selection, and Support Vector Machine (SVM) was employed for classification. The process involved examining unknown applications and extracting MD5 signatures, followed by feature extraction and selection phases. The combined static and dynamic analyses achieved a classification accuracy of 95% using the PCA-Relief method.

Nikola Milosevic et al. [11] focused on methods to identify malware on Android phones, which users often find difficult to detect. They implemented two approaches: one based on source code analysis and the other on permissions. Various machine learning methods such as Naive Bayes, SVM, C4.5 Decision Trees, and JRIP were used, with Weka tools facilitating the best performance through clustering techniques. Their ensemble learning approach, particularly SVM, achieved the highest precision of 95.8% and an F1-score of 95.6%.

Zhiwu Xu et al. [13] tackled malware issues on Android phones through three main tasks: graph extraction, graph encoding, and model training. Data was collected from Marvin and Contagio Dump, including malware and normal applications. The data was categorized into Control Flow Graph (CFG) and Data Flow Graph (DFG) forms and encoded into matrices for training deep learning models. The Convolutional Neural Network (CNN) approach, featuring reshaped layers and pooling, reduced training time and achieved high accuracy, with 99.8% in CDGDroid tools.

Burak TAHTACI et al. [14] reported the use of machine learning to detect Android malware. They developed models using n-gram properties of tiny files, focusing on feature selection and merging different extractions with trained models. Features with low selectivity and computation time were eliminated using variance threshold and information gain methods. This approach emphasized the creation of automated malware-scanning solutions.

Arvind Sangal et al. [15] developed 210 models using 21 different machine learning approaches and ten feature selection techniques. Their web-based framework, MLDroid, is aimed at detecting malware in Android apps. Utilizing 30 disparate datasets representing Android apps, the framework achieved an accuracy rate of 98.8% compared to various antivirus scanners. The study incorporated a substantial malware sample size of up to 55,000 and demonstrated superior performance using feature selection algorithms.

Xinning Wang et al. [17] proposed a multidimensional kernel functionality system and feature weight-based identification to classify and understand malicious and benign apps. Their approach utilized dynamic and static analyses, employing machine learning methods such as Naive Bayes, Neural Network, Decision Tree, and K-Nearest Neighbors. The TstructDroid Framework was suggested for investigating Android malware, founded on the dynamic study of kernel properties.

Anam Fatima et al. [19] presented a method for detecting new Android malware variants using machine learning in conjunction with static and dynamic analysis. They employed a Genetic Algorithm to optimize feature selection, reducing feature dimensionality by more than half. The study suggested using large datasets for better results and integrating Genetic Algorithms with other machine-learning techniques. Zhuo Ma et al. [20] introduced Droidetec, a deep neural network-based platform for static and automated Android malware detection. The platform employed a weight distribution mechanism to assess malware behavior sequences. Droidetec identified harmful code with a 95% success rate and an F1-score of 98.21%. The researchers utilized the AMD dataset, covering 65732 APIs with a mix of benign and malicious applications, and emphasized the integration with methods for identifying native shared libraries. ElMouatez Debbabi et al. [21] developed PetaDroid, a resilient and adaptable Android malware detection tool. The framework used CNN ensembles and confidence-based decision-making, aiming to address the evolving nature of Android APIs. PetaDroid outperformed MaMaDroid and MalDozer in various evaluation scenarios, suggesting the need for further real-world deployment validation. The study proposed expanding their work to include performance simulations for low-confidence detection, ensuring the dataset's proper division to avoid skewed findings.

In conclusion, these studies collectively highlight the evolution of machine learning approaches in Android malware detection, emphasizing the importance of feature selection, ensemble methods, and deep learning techniques in improving detection accuracy and robustness. Despite significant advancements in Android malware detection, several research gaps remain unaddressed. A major challenge lies in the detection of new and evolving malware variants, which traditional machine learning methods often fail to identify due to their reliance on predefined patterns and static features. Furthermore, many existing models struggle with large and high-dimensional datasets, resulting in diminished prediction accuracy and increased computational complexity. Additionally, while ensemble and deep learning techniques have shown promise, their application is often limited by issues such as overfitting, lack of real-time detection capabilities, and high training times.

The proposed Grid Tuned Two-Layered Stacking Approach for Malware Detection (GTTSAMD) aims to address these gaps by leveraging advanced stacking algorithms and grid search optimization. By integrating multiple base-level classifiers and a meta-classifier, our model enhances detection accuracy and robustness against dynamic malware threats. The use of repeated cross-validation and grid search ensures optimal hyperparameter tuning, reducing the risk of overfitting and improving generalization to new data.

Table 1. Analysis of existing systems for Android Malware detection

Author	Objective	Algorithm	Methodology Used	Merits	Demerits	Accuracy
Suleiman Y. Yerima et al.	To analyze malware-affected Androids	Machine Learning	Parallel classifications with rule-based, function-based, tree-based, and probabilistic methods	Parallel method facilitates easy value prediction	Unable to predict new malware	94.3%
Rishab Agrawal et al.	To identify new malware on Android phones	Semantic Analysis	Semantic and permission-based analysis using admin and user panels	In-depth semantic verification	Apk file upload varies between users	86.9%
Long Wen et al.	To detect malware using feature extraction techniques	SVM, PCA-Relief	Static and dynamic analysis, PCA for feature selection, SVM for classification	Effective malware detection using static and dynamic analysis	Difficult predictions with larger datasets	95.2%
Nikola Milosevic et al.	To enhance malware detection in Android phones	Machine Learning, SVM	Source code and permission-based analysis, ensemble learning	High precision achieved with SVM	Dynamic analysis may enhance performance	95.8%
Zhiwu Xu et al.	To improve malware detection using deep learning	CNN, CDGDroid	Extraction of data into CFG & DFG, training with CNN	Easy recognition of data through initial extraction into two parts	Training graph can be improved	93.7%
Burak TAHTACI	To detect Android malware using machine learning	PCA + SVM	Feature selection using variance threshold and information gain, SVM for classification	Reduced loss and utilization of high-dimensional data	It should include more features like n-gram frequencies, permissions, and threat intelligence	91.33%
Arvind Sangal	To develop a robust malware detection framework	MLDroid	Feature selection algorithms, machine learning models, antivirus comparison	Tested with 60 different antivirus software and on many instances	It does not provide real-time detection	92.8%
Xinning Wang	To classify malicious and benign apps	TstructDroid	Automatic data collection, kernel functionality analysis	Automatic data collection, multiple kernel analysis, and high-dimension reduction	Time delay and high cost	94.98%
Anam Fatima	To detect new Android malware variants	Genetic Algorithm + SVM/NN	Static and dynamic analysis, feature selection using Genetic Algorithm,	Optimized feature subset reduces training complexity	Working on small datasets takes more time	94%

			SVM/NN for classification			
Zhuo Ma	To automate Android malware detection	Droidetec	LSTM network for sequence processing, Skip-gram method	LSTM network for sequence processing and automated learning using the Skip-gram method	Complicated network, long learning time	92.22%
ElMouatez Billah Karbab	To create a resilient Android malware detection tool	PetaDroid	CNN ensembles, static analysis, confidence-based decision-making	Homogeneous clusters, adaptive and resilient	Not resistant to sophisticated obfuscation methods, cannot detect malware in downloads at runtime	91.15%

Specifically, our model addresses the following research gaps:

1. Detection of New Malware Variants: By combining various classifiers, our model adapts to new malware patterns and features, thereby improving the identification of previously unseen threats.
2. Handling High-Dimensional Data: The two-layered stacking approach, coupled with Principal Component Analysis (PCA) for feature selection, effectively manages high-dimensional datasets, enhancing prediction accuracy and computational efficiency[18].
3. Mitigating Overfitting: Repeated K-Fold cross-validation provides a more robust evaluation of model performance, mitigating the risk of overfitting and ensuring reliable detection across different data splits.
4. Real-Time Detection Capabilities: The ensemble nature of the proposed model, along with the use of CNNs and other efficient algorithms, facilitates faster training and prediction times, making real-time detection more feasible [19].

Our proposed GTTSAMD model thus represents a significant advancement in the field of Android malware detection, offering a comprehensive solution that addresses critical limitations of existing methodologies. Through rigorous evaluation and optimization, this model aims to set a new standard for accuracy, robustness, and efficiency in cybersecurity applications.

4. Proposed Methodology

The proposed model employs a sophisticated two-layered stacking approach, optimized using grid search techniques, to enhance the performance of machine learning classifiers in malware detection.

The following explanation provides a detailed breakdown of the methodology, supported by the attached block diagram.

4.1. Explanation of Block Diagram

The block diagram illustrates the entire process flow of the proposed two-layered stacking model with grid search optimization.

4.1.1. Training Data and Scalar Data

The process begins with preparing the training data, which is then scaled to ensure that all features are on a comparable scale. This standardization process adjusts the values of each attribute so that the mean is close to zero and the variance is one. Mathematically, let X be the matrix of input features where each row represents an observation, and each column represents a feature. The standardization formula is given by:

$$X'_{ij} = \frac{X_{ij}-\mu_j}{\sigma_j} \tag{3}$$

Where X'_{ij} is the scaled value of the feature j for observation i, μ_j is the mean of the feature j, and σ_j is the standard deviation of the feature j. This ensures that all features contribute equally to the model's performance.

4.1.2. Cross-Validation

Cross-validation is employed to evaluate the robust performance of the model. Cross-validation splits the scaled data into multiple subsets or folds. Each fold is used once as a validation set, while the remaining folds form the training set. This process is repeated multiple times, and the results are averaged to provide a stable assessment of the model's accuracy. In mathematical terms, for K-fold cross-validation [20], the data is divided into K subsets. For each fold k, the model is trained on $K-1$ folds and tested on the k-th fold. The performance metric (e.g., accuracy) is averaged over all K folds:

$$CV\_\text{score} = \frac{1}{K}\sum_{k-1}^{K} \text{score}_k \tag{4}$$

Where score $_k$ is the performance metric for the k-th fold.

4.1.3. Base-Level Models (Level-0)

The cross-validated training data is fed into the base-level models, which include tuned versions of Naive Bayes (NB)[21], K-Nearest Neighbors (KNN)[22], and Support Vector Machine (SVM)[23]. Each model's hyperparameters are optimized using grid search to ensure the best performance. The Naive Bayes model utilizes Bayes' theorem for probabilistic classification:

$$P(C \mid x) = \frac{P(x|C) \cdot P(C)}{P(x)} \quad (5)$$

Where the goal is to find the class C that maximizes $P(C \mid x)$. The K-Nearest Neighbors model classifies a data point based on the majority class among its k nearest neighbors, with the distance metric typically being the Euclidean distance:

$$d(x_i, x_j) = \sqrt{\sum_{l-1}^{m} (x_{il} - x_{jl})^2} \quad (6)$$

The Support Vector Machine finds the optimal hyperplane that separates different classes, solving the optimization problem:

$$\min_{\mathbf{w},b} \frac{1}{2} \| \mathbf{w} \|^2 \quad \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \forall i \quad (7)$$

4.1.4. Predicted Values

The base-level models generate predicted values for the testing data, which are then used as inputs for the next level of the stacking process. These predictions form the basis for further modeling at the meta-level.

4.1.5. Meta Data-1 and Meta-Level Models (Level-1)

The predicted values from the base-level models form Meta Data-1, which is input into the metalevel models. These include tuned versions of Random Forest (RF) [24] and AdaBoost (ADA)[25]. Random Forest is an ensemble method that constructs multiple decision trees and merges their outputs. Mathematically, Random Forest is represented as an ensemble of decision trees:

$$RF = \{T_1, T_2, \dots, T_n\} \quad (8)$$

Where each tree T_i is trained on a bootstrapped sample of the data, and the final prediction is the majority vote (for classification) or average (for regression) of the trees. AdaBoost, a boosting algorithm, adjusts the weights of misclassified instances to improve classification accuracy. Each classifier h_t focuses on the errors of the previous classifiers, and the final model is:

$$H(x) = \sum_{t-1}^{T} \alpha_t h_t(x) \quad (9)$$

Where α_t is the weight of the classifier h_t.

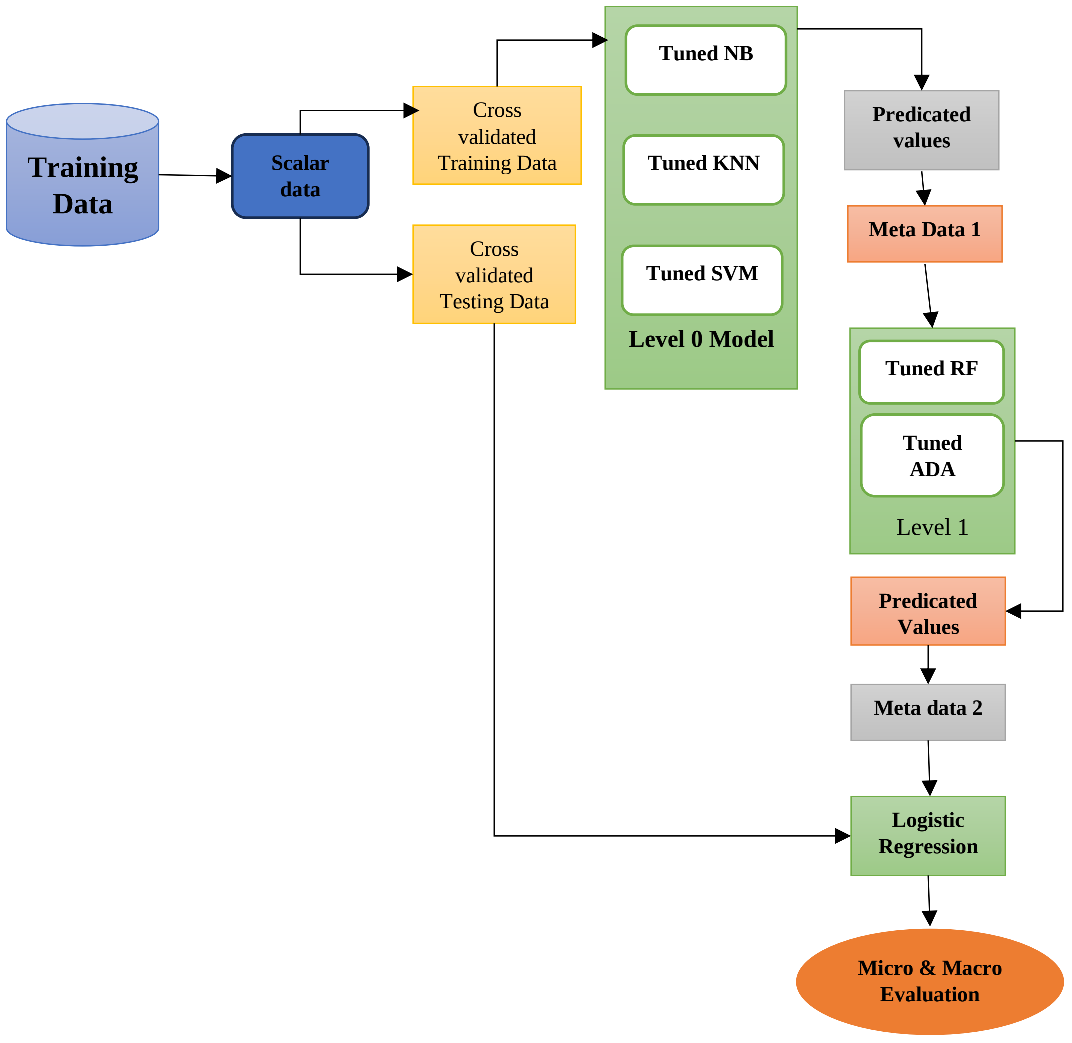

Fig. 3 Block diagram for two layered tuned stacking classifier

4.1.6. Predicted Values and Meta Data-2

The meta-level models generate their own predicted values, which form Meta Data-2. These values are further processed to refine the model's predictions.

4.1.7. Logistic Regression

Meta Data-2 is used as input for the final logistic regression model. Logistic regression is used to make the final prediction by combining the strengths of the base and meta-level models. The logistic regression model predicts the probability of the target variable using the sigmoid function:

$$P(y = 1 \mid x) = \frac{1}{1+e^{-(w\cdot x+b)}} \qquad (10)$$

Where the coefficients **w** and intercept b are learned using maximum likelihood estimation.

4.1.8. Micro and Macro Evaluation

The final predictions are evaluated using both micro and macro evaluation metrics to assess the model's performance comprehensively. The micro evaluation considers each individual prediction across all classes, while the macro evaluation averages the performance metrics of each class. Micro Precision is calculated as:

$$\text{Micro Precision } = \frac{\sum TP}{\sum TP+\sum FP} \qquad (11)$$

and Macro Precision is calculated as:

$$\text{Macro Precision } = \frac{1}{n}\sum_{i-1}^{n} \frac{TP_i}{TP_i+FP_i} \qquad (12)$$

Where TP is True Positives, FP is False Positives, and n is the number of classes. By following these steps, the model leverages cross-validation, multiple base-level models, and meta-level models to improve accuracy and robustness, evaluated comprehensively using micro and macro metrics.

The proposed model performs two-layered stacking by optimizing the parameters in the models. The algorithms used at the base predictor level are traditional tuned algorithms, but at level 1, the meta-classifiers are used as tuned ensemble algorithms. Here, the new predicted data exists in the form of numerical values. The proposed model has applied a logistic algorithm as the level 2 meta classifier. The model initially standardizes the data and then passes for the stacking classifier. The block diagram for the proposed model is projected in Figure 3.

4.2. Enhancing Performance of Base-Level Machine Learning Classifiers with Grid Search Optimization

The proposed model employs a sophisticated two-layered stacking approach, optimized using grid search techniques, to enhance the performance of machine learning classifiers in malware detection. The methodology is detailed below, explaining the process both theoretically and mathematically in an academic tone.

4.2.1. Grid Search Optimization Process

Definition of Hyperparameter Space

Identify key hyperparameters for each base-level algorithm. Define a range of possible values for each hyperparameter. For example: For K-Nearest Neighbors (KNN): Number of neighbors $k \in \{3,5,7,9\}$ For Support Vector Machine (SVM): Penalty parameter $C \in \{0.1,1,10\}$ and kernel type ∈ { linear, rbf }

Exhaustive Search

Perform an exhaustive search over all possible combinations of hyperparameters. For each combination, train and validate the classifier using cross-validation. Select the combination that yields the highest cross-validation accuracy.

Model Training with Optimal Hyperparameters

Retrain the classifier on the entire training dataset using the optimal hyperparameters identified in the exhaustive search.

Performance Evaluation

Evaluate the optimized classifier on a separate test dataset. Compute key performance metrics such as accuracy, precision, recall, and F1 score. Any machine learning model has two parameters: base parameters, which are fixed and tuning parameters, which are to be controlled and changed according to the model. If these parameters are not trained, the loss function will be maximum, and the misclassification rate will be high for the designed model. The tuning parameters update their values based on the training data available. These parameters are specific to the algorithm. There are different ways to optimize the parameters, as shown in Figure 4.

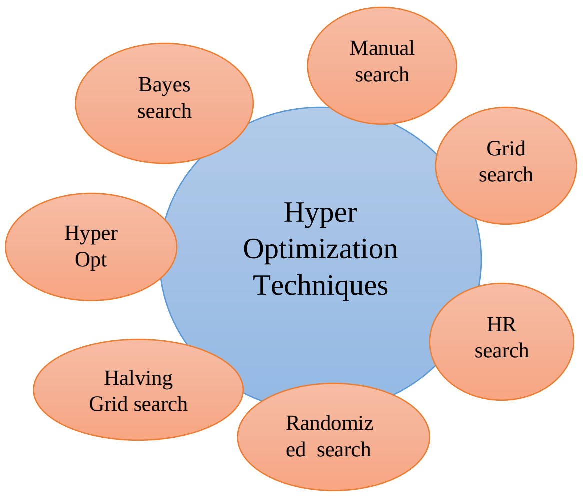

Fig. 4 Types of optimizers

The proposed model uses the Grid Search approach, which helps the model set its values before it trains the data. These are the crucial components of the algorithm, like the K value in KNN. Generally, this value is considered a random number between 3 to 10. The c value in SVM, which determines the penalty rate, generally lies between 0 and 1, but the larger C values help define a good marginal hyperplane. The advantage of this optimization technique lies in presenting accurate values by checking every possible combination of the algorithms. Random search chooses and tests an arbitrary combination of hyperparameters, whereas grid search examines every conceivable arrangement of hyperparameters to identify the optimal model. This method uses random sampling from a matrix of hyperparameters rather than a thorough search. In Grid Search, we test every possible combination of a predetermined list of hyper-parameter values and assess the model for each one. The design is comparable to a grid, where each value is organized into a matrix. The function is assessed at a range of arbitrary configurations in the dimensional space to optimize using random search. For smaller dimensional data, random search performs best because it takes fewer iterations and less time to identify the proper set. When there are fewer dimensions, a search algorithm is the optimal parameter search method. Grid search suffers from dimensionality issues when the range of hyperparameters increases exponentially, which is one of its main disadvantages. The process of tuning the algorithms using grid search is presented in Figure 5.

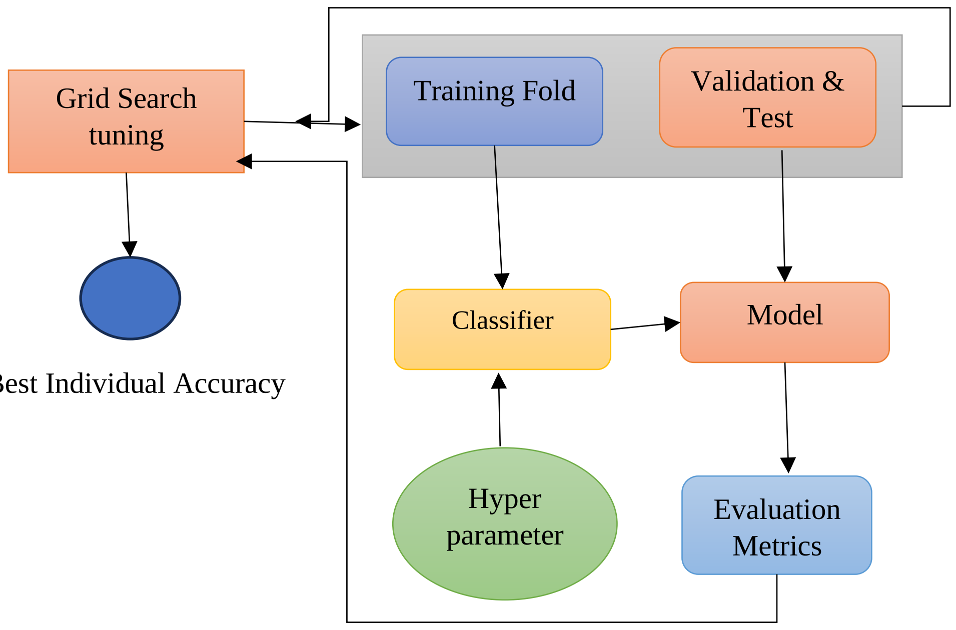

Fig. 5 Tuning using a grid search process

In the section below, the paper discusses the algorithm and its tuning parameters.

4.3. Tuning of Naïve Bayesian (Base Level)

The Naive Bayes algorithm, a supervised learning method based on Bayes' theorem, is widely used to address classification challenges. It is particularly effective for text categorization tasks and can handle large training sets efficiently.

The simplicity and effectiveness of the Naive Bayes[27] method make it one of the most straightforward machine learning algorithms for constructing reliable predictive models. As a probabilistic classifier, Naive Bayes makes decisions based on the likelihood of specific events.

It is known for its rapid classification capabilities and is suitable for both binary and multiclass categorization problems. The algorithm performs exceptionally well in multiclass predictions compared to other algorithms, making it the preferred choice for text categorization problems. The Bernoulli Naive Bayes [27] variant is particularly useful for binary/boolean features. The classification decision is based on the following equation:(13)

$$Y_{pred} = max_y * P(Y) * \prod_{i=0}^{n} P(F_i|Y)- \qquad (13)$$

Where:

- $P(Y)$ is the prior probability of class Y,
- $P(F_i \mid Y)$ is the likelihood of a feature F_i given class Y,
- $\hat{Y}$ is the predicted class.

By tuning these parameters, the Naive Bayes model can be optimized to achieve higher accuracy and robustness in classification tasks, particularly in scenarios involving large and diverse datasets.

This optimization process is integral to ensuring that the Naive Bayes algorithm can effectively differentiate between various classes, thereby enhancing its utility in practical applications.

Table 2. Tuning parameters of Bernoulli Naive Bayes approach

Parameter	Description	Possible Values
Binarize	This setting is only relevant for the Bernoulli Naive Bayes methodology. It defines a cutoff value for binarizing sample characteristics. If sample features are already binarized, this parameter can be omitted.	Floating value to define the threshold
Priors	Specifies the prior class probabilities. When priors are provided (in an array), they remain unchanged regardless of the dataset.	Number of class labels
Alpha	The smoothing parameter α\alphaα for Laplace or Lidstone smoothing, which different Naive Bayes algorithms might use to handle zero probabilities.	Any floating value between 0 and 1

Table 3. Tuning parameters of the KNN algorithm

Parameter	Description	Possible Values
n_neighbors	Specifies the number of nearest neighbors to consider. Optimal kkk values are typically small prime numbers to balance bias and variance.	Any prime number
Weights	Determines the influence of each neighbor on the classification. "Uniform" gives equal weight to all neighbors, while "distance" weighs points according to the reciprocal of their distances.	Uniform, Distance

4.4. Tuning of K-Nearest Neighbors (Base Level)

The K-Nearest Neighbors (KNN) algorithm is a straightforward yet powerful classification technique that assigns observations to the class of their k-nearest neighbors. It operates by identifying the k-nearest data points to a given observation based on a specified distance metric.

Fine-tuning the hyperparameters of KNN is crucial for optimizing its performance, particularly in tasks like malware detection [28].

Key Tuning Parameters for KNN

- Number of Neighbors (k): The parameter k represents the number of nearest neighbors considered for classification. Choosing the appropriate value for k is critical. Small values of k make the model sensitive to noise and prone to overfitting, while larger values can lead to oversimplification and underfitting. A typical range for k is between 3 and 10.
- Distance Metric: KNN uses a distance metric to measure the similarity between data points. The default distance metric in many implementations, including scikit-learn, is Euclidean distance. However, other metrics such as Manhattan distance or cosine distance can also be used depending on the nature of the data and the specific application.
- Weight Function: The algorithm can classify based on a simple majority vote or weigh the neighbors' votes according to their distance from the observation being classified. Weight functions such as inverse distance weighting or kernel density weighting can be employed. Using weights can improve classification accuracy by giving more influence to closer neighbors.
- Preprocessing: KNN is sensitive to the scale and distribution of input data. Therefore, it is essential to preprocess the data to ensure all features are on a similar scale and distribution. Techniques like normalization or standardization are commonly used to prepare the data for KNN.

4.4.1. Mathematical Framework

Given an observation x, the classification decision in KNN can be mathematically expressed as:

$$y = \arg\max_{c} \sum_{i \in N_k} w_i \cdot \delta(y_i, c) \tag{14}$$

Where:

- N_k represents the set of k -nearest neighbors,
- w_i is the weight assigned to neighbor i based on its distance,
- $\delta(y_i, c)$ is the Kronecker delta function, which is 1 if the class of neighbor i is c and 0 otherwise.

By optimizing these parameters through grid search, the KNN algorithm can be fine-tuned to achieve superior performance in classification tasks. This optimization ensures that the KNN model effectively captures the underlying patterns in the data, leading to more accurate and reliable predictions. In summary, tuning the hyperparameters of KNN involves selecting the optimal number of neighbors, choosing the appropriate distance metric, and deciding on the weight function. Preprocessing the data is also essential to ensure the model performs well. By carefully adjusting these parameters, the KNN algorithm can be tailored to provide high accuracy and robustness.

4.5. Tuning of SVM (Base Level)

Support Vector Machine (SVM) is a powerful classification method that works by finding the optimal hyperplane to separate data points into different classes. The tuning of SVM parameters is crucial to achieving high classification performance, particularly when dealing with complex datasets such as those used in malware detection.

4.5.1. Key Tuning Parameters for SVM

- Penalty Parameter (C): The parameter C controls the trade-off between achieving a low error on the training data and minimizing the complexity of the model. Higher values of C put more emphasis on classifying all training examples correctly, which can lead to overfitting. Lower values of C allow the margin to be maximized at the cost of more training errors. The typical values for C are multiples of 10, starting from 0.1.

- Gamma (γ): The parameter γ is relevant for non-linear SVMs such as the Radial Basis Function (RBF) SVM. It defines how far the influence of a single training example reaches. Low values of γ imply a large radius of influence, resulting in a smoother decision boundary. High values of γ imply a smaller radius of influence, resulting in a more complex decision boundary that may overfit the training data. Typical values for γ are powers of 10, starting from 0.1.
- Kernel Type: The kernel function transforms the input data into the required format. SVMs can use various kernel functions such as linear, polynomial, and RBF. The choice of kernel affects the performance of the SVM, and it is essential to experiment with different kernels to find the one that best fits the data.
- Probability Estimates: Enabling probability estimates allows the model to output the probability that a data point belongs to a particular class rather than just the predicted class label. This is useful for applications requiring a measure of confidence in the model's predictions. Enabling this feature involves additional computation as it uses internal cross-validation.

Table 4. SVM tuning parameters

Parameter	Description	Possible Values
C	The penalty parameter determines the trade-off between classification accuracy on the training data and the complexity of the decision boundary. Higher values prevent misclassification.	Multiples of 10 starting from 0.1
Gamma	Defines the influence of a single training example. Low values signify "far" reach, and high values signify "near" reach. Higher values can lead to overfitting.	Powers of 10 starting from 0.1
Kernel	Specifies the kernel function to be used in the algorithm. Different kernels transform the data in various ways to find the optimal hyperplane.	Linear, polynomial, RBF, etc.
Probability	Determines whether to enable probability estimates. Enabling this option can provide additional insights into the confidence of predictions but requires more computation.	Boolean value (True/False)

Algorithm for Two-Layered Stacking - Voting Classifier: The following algorithm outlines the process of implementing a two-layered stacking classifier with optimized base and meta-level models.

Input: Load the Malware Dataset, “AMData”

Output: Metric Analysis

```
Begin:
Step 1. An MData ←Load Malware Dataset
Step 2. for i in len(AMData):
    if(AMData[i].type==”String”):
        AMData[i] ←label_encoder.transform(AMData[i])
    if(AMData[i].type==”int”):
            new_value[i] ← $\frac{\sum_{i=0}^{len(AMData)} AMDATA[i]-\mu}{\sigma^2}$
            AMData[i] ←new_value[i]
Step 3 . a. AMData_new, AMData_new_test ←RepeatCV(folds=5)
Step 4. base_estimators=[]
Step 5. define parameters for BerunoliBN, KNN, SVM and fit grid search
Step 6. compute best parameters and best score and append to base_estimators
Step 7. for i in len(base_estimators)
    for j in base_estimators[i]
class ←predict(base_estimator[i].fit(AMData_new[i][j]))
Step 8. meta_estimators=[]
Step 9. define parameters for Random Forest, ADA and fit to grid search
Step 10. compute best parameters, best score and append to meta_estimators
Step 11. for i in len(meta_estimators)
    for j in meta_estimators[i]
class_pred ←predict(meta_estimator[i].fit(AMData_new_test[i][j]
Step 12. final_class ←StackingClassifier(meta_estimator,Tuned_LogRegression)
Step 13. print metrics
End
```

This detailed explanation and algorithm outline how to effectively tune and implement a Support Vector Machine (SVM) as part of a two-layered stacking model for malware detection. By optimizing the key parameters, the SVM can be tailored to provide high accuracy and reliability, enhancing the overall performance of the detection system.

4.6. Meta Classifier

A meta-classifier is an advanced meta-learning algorithm used in classification predictive modeling tasks. It leverages the predictions of multiple base classifiers to generate a final prediction, combining their strengths to improve overall accuracy. In the stacking ensemble method, the meta-classifier uses the outputs from several base models as input features to produce the final classification.

In a two-layered stacking framework, the first layer consists of multiple base classifiers, each trained on the entire training dataset. Their predictions are then used as input for the second layer, where the meta-classifier (or regression model) processes these predictions to generate the final output. This hierarchical approach enhances the model's ability to generalize by integrating diverse

4.6.1. Tuning of Random Forest (Meta Level-1)

The Random Forest algorithm, an ensemble technique, constructs a multitude of decision trees during training. Each tree in the forest votes for a class and the class with the most votes is chosen as the final prediction. This method leverages the power of multiple uncorrelated models to enhance prediction accuracy and robustness. The principle behind Random Forest is that combining the predictions from multiple trees reduces the risk of overfitting.

Key Tuning Parameters for Random Forest

- max_features: This parameter determines the maximum number of features Random Forest is allowed to consider when splitting a node. Using 'auto' or 'sqrt' is common to avoid overfitting.
- n_estimators: The number of trees in the forest. More trees generally improve performance but also increase computational cost. Values are typically multiples of 10.
- min_sample_leaf: The minimum number of samples required to be at a leaf node. Smaller values can lead to a model that captures noise in the data. Powers of 2 are commonly used.
- max_depth: The maximum depth of the tree. Limiting the depth prevents overfitting. Integer values with an interval of 2 are typical.
- min_sample_split: The minimum number of samples required to split an internal node. This parameter helps to control the growth of the tree and prevent overfitting. Values are usually multiples of 5.

4.6.2. Tunning of Ada Boost (Meta Level-1)

AdaBoost, or Adaptive Boosting, is an ensemble technique that adjusts the weights of incorrectly classified instances, giving them more emphasis in subsequent iterations.

This method aims to convert weak learners into strong ones by focusing more on the challenging cases. AdaBoost is particularly effective in reducing bias and variance in supervised learning.

Key Tuning Parameters for AdaBoost

- base_estimator: Specifies the type of weak learner used. Common choices include decision trees, logistic regression, and Support Vector Classifiers (SVC).
- n_estimators: Number of weak learners to use. More estimators generally improve performance up to a point. The default value is 50, but multiples of 10 are often used.
- learning_rate: Shrinks the contribution of each weak learner. A smaller value requires more weak learners to maintain performance. Typical values range from 0 to 1, with a fine-tuning interval.

4.6.3. Designing the Meta Classifier 2 using Tuned Logistic Regression

Logistic regression is a widely used classification algorithm that predicts the probability of an observation belonging to a certain class. It is particularly effective for binary classification problems. The model is trained using a labeled dataset with a binary output variable.

Table 5. Random forest tuning parameters

Parameter	Description	Possible Values
max_features	Maximum number of features considered for splitting a node.	Auto or sqrt
n_estimators	Number of trees in the forest. More trees provide better performance but increase computation time.	Multiples of 10
min_sample_leaf	Minimum number of samples is required to be at a leaf node. Smaller values may lead to overfitting.	Powers of 2
max_depth	Maximum depth of the tree. Limiting depth helps prevent overfitting.	Integer values with an interval of 2
min_sample_split	Minimum number of samples required to split an internal node. Controls tree growth.	Multiples of 5

Table 6. AdaBoost tuning parameters

Parameter	Description	Possible Values
base_estimator	Type of weak learners to use (e.g., decision tree, logistic regression).	Any traditional ML algorithm
n_estimators	Number of weak learners. More learners can improve performance but increase computational cost.	Multiples of 10
learning_rate	Reduces the contribution of each weak learner to avoid overfitting.	0 to 1, with fine-tuning intervals

Key Tuning Parameters for Logistic Regression

- C: The regularization parameter. Higher values reduce regularization, emphasizing fitting the training data. Lower values increase regularization, focusing on simplicity.
- Penalty: Specifies the type of regularization. L1 (lasso) regularization can lead to sparse models, while L2 (ridge) regularization tends to spread the error across all parameters.
- Dual: Indicates whether to use the dual formulation, applicable for L2 penalty only. This is useful for large datasets with many features.
- Solver: Optimization algorithm to use to fit the model. Choices include 'liblinear', 'sag', 'saga', etc., each suited for different types of datasets and objectives.

Table 7. Logistic Regression tuning parameters

Parameter	Description	Possible Values
C	Regularization parameter. High values reduce regularization, and low values increase it.	Logarithmic values
Penalty	Type of regularization (L1 or L2). L1 for lasso, L2 for ridge.	L1, L2 regularizations
Dual	Dual or primal formulation. Dual is used for L2 penalty with 'liblinear' solver.	Boolean
Solver	Optimization algorithm for fitting the model. Suitable choices depend on dataset size and type.	'liblinear', 'sag', 'saga', etc.

By carefully tuning these parameters, the logistic regression model can achieve high accuracy and reliability in classification tasks, contributing significantly to the overall performance of the two-layered stacking model.

5. Results & Discussion

5.1. System Setup and Implementation Details

The system setup involves a comprehensive approach to optimize and evaluate the performance of machine learning models for malware detection. The implementation utilizes a two-layered stacking ensemble method, combining various base and meta classifiers tuned using grid search optimization. The primary components of the system setup include data preprocessing, model training, and evaluation.

5.1.1. Data Preprocessing

The dataset undergoes normalization to scale the values of independent attributes. This process accelerates computations, particularly in the AdaBoost meta-classification, by maintaining variance values close to 1. Normalization ensures that features are scattered around the nearest points, facilitating faster and more accurate learning.

5.1.2. Model Training and Hyperparameter Tuning

Various machine learning algorithms are tuned using the grid search approach to find the best hyperparameters that maximize model accuracy. The algorithms include Naive Bayes, K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Random Forest, AdaBoost, and Logistic Regression.

5.1.3. Dataset Used

The dataset used in this research is the DroidFusion on the Drebin-215 dataset,[29] which contains 215 attributes representing various features extracted from Android applications, such as API call signatures, permissions, and other relevant characteristics.

This dataset, sourced from the public repository Kaggle, includes 15,036 samples, comprising 5,560 malware and 9,476 benign applications.

The feature categories are detailed in an accompanying file, which classifies the attributes into specific groups. This dataset serves as the foundation for evaluating the effectiveness of feature selection techniques in enhancing malware detection accuracy on Android devices.

5.2. Evaluation and Performance Analysis

The performance of the proposed two-layered stacking model for Android malware detection was evaluated using a confusion matrix, as depicted in the heatmap (Figure 6).

The dataset used for this evaluation was a reduced subset, comprising 0.452% of the original data, to facilitate more manageable computation.

Original dataset:

- Total samples: 15,036
- Malware samples: 5,560
- Benign samples: 9,476

Confusion matrix: Total samples represented: 68

The compression percentage is calculated as:

$$\text{Compression Percentage} = \left(\frac{\text{Total samples in confusion matrix}}{\text{Total samples in original dataset}}\right) \times 100$$

Using the given numbers:

$$\text{Compression Percentage} = \left(\frac{68}{15036}\right) \times 100 \approx 0.452\%$$

Thus, the confusion matrix represents approximately 0.452% of the original dataset.

The confusion matrix provides a detailed breakdown of the model's classification results. It includes the following metrics:

- True Positives (TP): Instances where malware was correctly identified as malware.
- True Negatives (TN): Instances where benign applications were correctly identified as benign.
- False Positives (FP): Instances where benign applications were incorrectly classified as malware.
- False Negatives (FN): Instances where malware was incorrectly classified as benign.

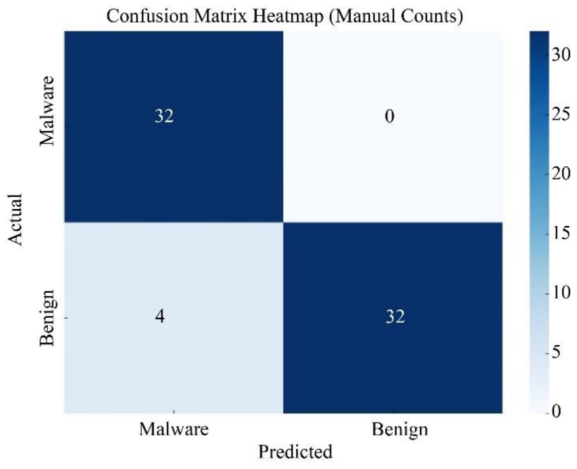

Fig. 6 Confusion matrix heatmap for reduced dataset

The heatmap illustrates these values, with the x-axis representing the predicted classifications and the y-axis representing the actual classifications. The diagonal elements (TP and TN) indicate correct classifications, while the off-diagonal elements (FP and FN) represent misclassifications.

This confusion matrix is essential for assessing the classification accuracy and the ability of the proposed model to distinguish between benign and malicious applications effectively. The visual representation in the heatmap highlights the model's performance, indicating a high level of accuracy with minimal misclassifications, thereby validating the efficacy of the two-layered stacking approach in Android malware detection.

5.2.1. Hyperparameter Tuning Configuration

The grid search [30] process involves specifying a range of values for each hyperparameter and evaluating all possible combinations to identify the optimal configuration. Table 8 summarizes the tuning parameters and the best values obtained through this process, along with their respective accuracies. “Fits” refers to the number of times the model is trained and evaluated during the hyperparameter tuning process. When using grid search or other hyperparameter optimization techniques, the algorithm evaluates various combinations of hyperparameters to determine the best set of parameters for the model. Each combination of hyperparameters represents a candidate configuration, and the model is "fit" to the training data using each configuration. Therefore, "Total Number of Fits" indicates the total number of times the model was trained and evaluated with different combinations of hyperparameters during the tuning process. This number is the product of the number of candidates (combinations of hyperparameters) and the number of cross-validation splits used in the tuning process.

Figure 7 illustrates the accuracy of the standalone algorithms after tuning. The X-axis represents the algorithm names, and the Y-axis denotes the accuracy. Among the six models, SVM and Logistic Regression achieved the highest accuracy, both reaching 95.4%

Table 8. Accuracy analysis using grid parameters

Algorithm Name	Number of Tuning Parameters	Total Number of Fits	Best Values	Accuracy
Naive Bayesian	3	40 Candidates & 200 Fits	{'alpha': 0.01, 'binarize': 0.0, 'fit_prior': True}	85.1%
KNN	4	480 Candidates & 2400 Fits	{'metric': 'minkowski', 'n_neighbors': 13, 'p': 1, 'weights': 'distance'}	94.03%
SVM	4	200 Candidates & 1000 Fits	{'C': 100, 'gamma': 0.0001, 'kernel': 'sigmoid', 'probability': True}	95.4%
Random Forest	5	720 Candidates & 3600 Fits	{'max_depth': 10, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 1, 'n_estimators': 28}	94.3%
AdaBoost	3	320 Candidates & 1600 Fits	{'estimator': DecisionTreeClassifier(), 'learning_rate': 1.02, 'n_estimators': 28}	90.29%
Logistic Regression	4	168 Candidates & 840 Fits	{'C': 0.1, 'dual': False, 'penalty': 'l2', 'solver': 'lbfgs'}	95.4%

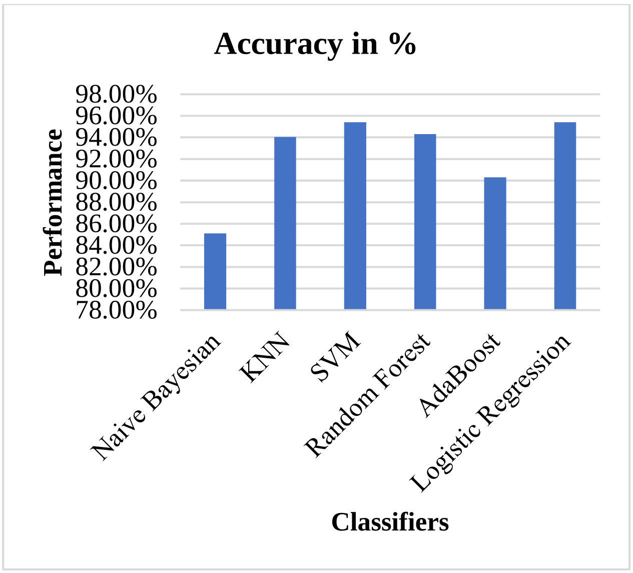

Fig. 7 Accuracy of different algorithms with tuned parameters

The proposed two-layered stacking model, which integrates these algorithms, achieved an overall accuracy of 96%, as shown in Figure 7. This combined approach surpasses the performance of standalone models, demonstrating enhanced accuracy, recall, and precision.

Table 9. Confusion matrix for Malware analysis

Actual	Predicted	Count
Malware	Malware (TP)	32
Malware	Benign (FN)	4
Benign	Malware (FP)	0
Benign	Benign (TN)	32
Total		68

True Positive and False Positive Rates

True Positive and False Positive Rates

The true positive malware rate is calculated as follows:

$$\text{True Positive Malware Rate} = \frac{\text{TP}}{\text{TP+FN}} = \frac{32}{32+4} = 0.89$$

The false positive malware rate is calculated as follows:

$$\text{False Positive Malware Rate} = \frac{\text{FP}}{\text{FP+TN}} = \frac{0}{0+32} = 0$$

Explanation: The confusion matrix provides a summary of the classification performance of the proposed model on the malware dataset. It categorizes the predictions into four types:

- True Positive (TP) Malware: Correctly identified malware instances (32).
- False Negative (FN) Malware: Malware instances that were incorrectly classified as benign (4).
- True Negative (TN) Malware: Correctly identified benign instances (32).
- False Positive (FP) Malware: Benign instances that were incorrectly classified as malware (0).

The true positive rate of 0.89 indicates that the model correctly identified 89% of the actual malware instances. The false positive rate of 0 indicates that no benign instances were misclassified as malware, demonstrating the model's high precision in identifying benign applications.

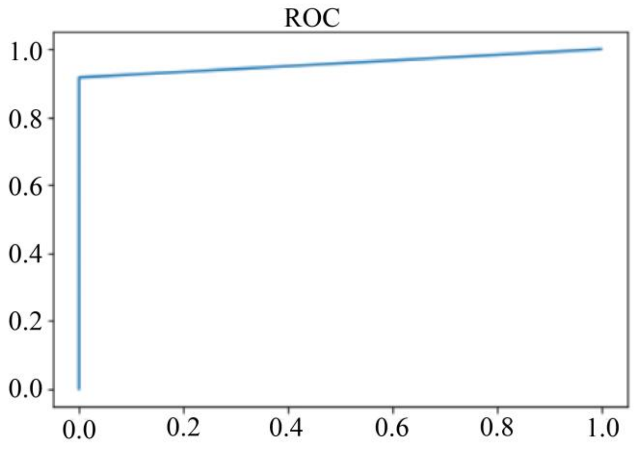

Fig. 8 ROC curve analysis

Figure 8 presents the Receiver Operating Characteristics (RoC) curve of the proposed model, comparing true and false positive rates across different threshold points. The X-axis denotes the false positive rate, while the Y-axis denotes the true positive rate computed from the confusion matrix presented in Table 9.

5.3. Findings of the Study

The study's findings provide a comprehensive analysis of the performance of various machine learning algorithms in malware detection, with a focus on the optimization of these algorithms through grid search. The Naive Bayesian algorithm achieved an accuracy of 85.1%, demonstrating moderate effectiveness, while K-Nearest Neighbors (KNN) reached 94.03% accuracy, highlighting its efficacy in distinguishing between benign and malicious applications. Support Vector Machine (SVM) and Logistic Regression both achieved impressive accuracies of 95.4%, underscoring their robustness in handling complex classification tasks when appropriately tuned. Random Forest, with an accuracy of 94.3%, showed strong performance due to its ability to handle complex interactions and many features, whereas AdaBoost attained an accuracy of 90.29%, indicating its utility in enhancing weak learners' performance through iterative boosting. Notably, the combined approach using a two-layered stacking ensemble model achieved the highest overall accuracy of 96%, surpassing individual models and demonstrating significant improvements in recall and precision rates, with a 100% recall rate for class 0 labels and a 100% precision rate for class 1 labels. These results underscore the importance of hyperparameter tuning and ensemble methods in optimizing model performance, offering valuable insights for developing robust and accurate malware detection systems in the field of cybersecurity.

5.4. Limitations of the Study and Future Suggestions

Despite the promising results, the study has several limitations that need to be addressed in future research. One key limitation is the reliance on a specific dataset with 215 attributes, which may not capture the full diversity of malware behaviors and characteristics present in real-world scenarios. This could limit the generalizability of the findings to other datasets and environments. Additionally, while the grid search optimization process was effective in enhancing model performance, it is computationally intensive. It may not be feasible for extremely large datasets or in resource-constrained settings. The study also primarily focused on the accuracy of the models, with less emphasis on other important metrics such as training time, model interpretability, and computational efficiency. Furthermore, the combined stacking model, although achieving high accuracy, introduces additional complexity and requires careful calibration to prevent overfitting and ensure robustness.

For future research, it is recommended to explore more diverse and representative datasets to validate and generalize the findings across different contexts. Incorporating real-time data and evolving malware patterns could provide a more comprehensive assessment of model performance. Additionally, exploring alternative optimization techniques, such as Bayesian optimization or evolutionary algorithms, could reduce computational overhead while still effectively tuning hyperparameters. Future studies should also consider a broader range of evaluation metrics, including training time, computational cost, and model interpretability, to provide a more holistic evaluation of model performance. Finally, developing lightweight and scalable ensemble methods could enhance the practicality of the proposed approaches for real-world applications, particularly in environments with limited computational resources. These future directions will help refine the methodologies and improve the robustness and applicability of machine learning models for malware detection.

6. Conclusion

The proposed model mainly focuses on dynamic attributes. So, it generates the subset from the 215 attributes based on the type of malware using the Integrated RFE and SVM approach. For these attributes, the model tries to combine multiple models, which are tuned using the Grid Search approach. Grid search always checks every possibility of the combination with all the unique values assigned to the estimator. The major goal of this research is to predict different types of malware without taking a single chance to misclassify the data and try to train the data with more records in the form of meta data instead of a few test datasets. The prediction levels are assigned a probability of both correct and incorrect classification rate, i.e., suppose the model has applied three traditional tuned approaches at the base level. It generates 8 possibilities by considering all models that have predicted true, all that have predicted wrong and other cases and considers the majority voted results as prediction output. In this way, the proposed research analyses every possibility in terms of both estimator and models and enhances the performance.

References

[1] Ivan Dychka et al., “Malware Detection Using Artificial Neural Networks,” Advances in Computer Science for Engineering and Education II, Advances in Intelligent Systems and Computing, vol. 938, pp. 3-12, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[2] Zarni Aung, and Win Zaw, “Permission-Based Android Malware Detection,” *International Journal of Scientific & Technology Research*, vol. 2, no. 3, pp. 228-234, 2018. [Google Scholar]

[3] İsmail Atacak, Kazım Kılıç, and İbrahim Alper Doğru, “Android Malware Detection Using Hybrid ANFIS Architecture with Low Computational Cost Convolutional Layers,” *PeerJ Computer Science*, vol. 8, pp. 1-23, 2022. [CrossRef] [Google Scholar] [Publisher Link]

[4] Imtiyaz Khan et al., “Secure and Efficient Data Sharing Scheme for Multi-User and Multi-Owner Scenario in Federated Cloud Computing,” *Journal of Theoretical and Applied Information Technology*, vol. 102, no. 6, pp. 2541-2555, 2024. [Google Scholar] [Publisher Link]

[5] Min Zhao et al., “AntiMalDroid: An Efficient SVM-Based Malware Detection Framework for Android,” *Information Computing and Applications*, Communications in Computer and Information Science, vol. 243, pp. 158-166, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[6] Kuruva Laxmanna, K. Lakshmi, and S. Prem Kumar, “Identifying Malwares by Signature Distribution Algorithm in MANET with Assorted Strategy,” *International Journal of Computer Engineering in Research Trends*, vol. 2, no. 9, pp. 636-639, 2015. [Google Scholar] [Publisher Link]

[7] Rodney Anthony Raj, and A.R. Chayapathi, “Malware as a Component in Cybercrime: A Survey,” *International Journal of Computer Engineering in Research Trends*, vol. 4, no. 5, pp. 176-179, 2017. [Google Scholar] [Publisher Link]

[8] Suleiman Y. Yerima, and Sakir Sezer, “DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection,” *IEEE Transactions on Cybernetics*, vol. 49, no. 2, pp. 453-466, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[9] Rishab Agrawal et al., “Android Malware Detection Using Machine Learning,” *International Conference on Emerging Trends in Information Technology and Engineering*, Vellore, India, pp. 1-4, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[10] Long Wen, and Haiyang Yu, “An Android Malware Detection System Based on Machine Learning,” *AIP Conference Proceedings*, vol. 1864, no. 1, pp. 1-7, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[11] Nikola Milosevic et al., “Machine Learning Aided Android Malware Classification,” *Computers & Electrical Engineering*, vol. 61, pp. 266-274, 2017. [CrossRef] [Google Scholar] [Publisher Link]

[12] K. Thejeswari, K. Sreenivasulu, and B. Sowjanya, “Cyber Threat Security System Using Artificial Intelligence for Android-Operated Mobile Devices,” *International Journal of Computer Engineering in Research Trends*, vol. 9, no. 12, pp. 275-280, 2022. [CrossRef] [Publisher Link]

[13] Zhiwu Xu et al., “CDGDroid: Android Malware Detection Based on Deep Learning using CFG and DFG,” *Formal Methods and Software Engineering*, vol. 11232, pp. 177-193, 2018. [CrossRef] [Google Scholar] [Publisher Link]

[14] Burak Tahtaci, and Beyzanur Canbay, “Android Malware Detection Using Machine Learning,” *Innovations in Intelligent Systems and Applications Conference*, Istanbul, Turkey, pp. 1-6, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[15] Arvind Mahindru, and A.L. Sangal, “MLDroid—Framework for Android Malware Detection using Machine Learning Techniques,” *Neural Computing and Applications*, vol. 33, pp. 5183-5240, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[16] T. Monisha, R. Sridevi, and K.R. Tirumalini, “Detection of Malicious URLs Using Artificial Intelligence,” *International Journal of Computer Engineering in Research Trends*, vol. 7, no. 8, pp. 6-10, 2020. [Publisher Link]

[17] Xinning Wang, and Chong Li, “Android Malware Detection through Machine Learning on Kernel Task Structures,” *Neurocomputing*, vol. 435, pp. 126-150, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[18] Rodney Anthony Raj, and A.R. Chayapathi, “A Honeypot for a Small Network using Raspberry Pi,” *International Journal of Computer Engineering in Research Trends*, vol. 4, no. 8, pp. 319-324, 2017. [Publisher Link]

[19] B. Prasanthi, Suresh Pabboju, and D. Vasumathi, “Query Adaptive Hash-Based Image Retrieval in Intent Image Search,” *Journal of Theoretical & Applied Information Technology*, vol. 93, no. 2, pp. 278-286, 2016. [Google Scholar] [Publisher Link]

[20] Zhuo Ma et al., “Droidetec: Android Malware Detection and Malicious Code Localization through Deep Learning,” *Arxiv*, pp. 1-13, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[21] ElMouatez Billah Karbab, and Mourad Debbabi, “PetaDroid: Adaptive Android Malware Detection Using Deep Learning,” *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, Cham: Springer, pp. 319-340, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[22] Vasileios Kouliaridis, and Georgios Kambourakis, “A Comprehensive Survey on Machine Learning Techniques for Android Malware Detection,” *Information*, vol. 12, no. 5, pp. 1-12, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[23] Mohammed K. Alzaylaee, Suleiman Y. Yerima, and Sakir Sezer, “DL-Droid: Deep Learning-Based Android Malware Detection using Real Devices,” *Computers & Security*, vol. 89, pp. 1-11, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[24] Tianliang Lu et al., “Android Malware Detection Based on a Hybrid Deep Learning Model,” *Security Communication Networks*, vol. 2020, no. 1, pp. 1-11, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[25] Omar N. Elayan, and Ahmad M. Mustafa, “Android Malware Detection Using Deep Learning,” *Procedia Computer Science*, vol. 184, pp. 847-852, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[26] Halil Murat Ünver, and Khaled Bakour, “Android Malware Detection Based on Image-Based Features and Machine Learning Techniques,” *SN Applied Sciences*, vol. 2, pp. 1-15, 2020. [CrossRef] [Google Scholar] [Publisher Link]

[27] Stuart Millar et al., “Multi-View Deep Learning for Zero-Day Android Malware Detection,” *Journal of Information Security and Applications*, vol. 58, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[28] Nadia Daoudi et al., “Lessons Learnt on Reproducibility in Machine Learning Based Android Malware Detection,” *Empirical Software Engineering*, vol. 26, pp. 1-53, 2021. [CrossRef] [Google Scholar] [Publisher Link]

[29] Marília Prata, Android Malware LSTM, Kaggle, 2021. [Online]. Available: https://www.kaggle.com/code/mpwolke/android-malware-lstm/input

[30] Tsehay Admassu Assegie, “An Optimized KNN Model for Signature-Based Malware Detection,” *International Journal of Computer Engineering in Research Trends*, vol. 8, no. 2, pp. 46-49, 2021. [CrossRef] [Google Scholar] [Publisher Link]