Journal of Theoretical and Applied Information Technology
15th February 2024. Vol.102. No 3
© Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

# ENHANCING ANDROID MALWARE DETECTION THROUGH ENSEMBLE STACKING CLASSIFIERS AND REGULARIZATION-BASED FEATURE SELECTION

**ELISA BAHARA SORITUA[1] , DITDIT NUGERAHA UTAMA[2]**
[1]Student, Master Program in Computer Science, BINUS Graduate Program, BINUS University, Indonesia
[2]Lecturer, Master Program in Computer Science, BINUS Graduate Program, BINUS University, Indonesia
E-mail: [1]elisa.soritua@binus.ac.id, [2]ditdit.utama@binus.ac.id

**ABSTRACT**

With the persistent evolution of Android malware, advanced detection techniques have become paramount. This paper introduces a novel approach to Android malware detection, harnessing the power of ensemble stacking classifiers combined with regularization-based feature selection techniques, specifically Lasso, Ridge, and Elastic Net, applied to the Drebin-215 dataset. Using base classifiers including Random Forest, K-Nearest Neighbors, Support Vector Machine, Logistic Regression, and Bernoulli Naive Bayes, with Logistic Regression as the meta classifier, our methodology aims to capture the collective strengths of diverse algorithms. Initial results of individual classifiers laid the foundation, upon which the ensemble model furthered the detection rates. Implementing the regularization-based feature selection significantly enhanced the model's efficiency, leading to improved classification accuracy. Compared to traditional methods, our proposed system offers a notable enhancement in malware detection capabilities, providing a resilient solution to the prevailing challenges in Android security. This study underscores the potential of integrating machine learning-driven ensembles with advanced feature selection techniques for bolstered security measures.

**Keywords:** *Android Malware Detection, Machine Learning, Ensemble Stacking, Genetic Algorithm.*

## 1. INTRODUCTION

Android is one of the most widely used operating systems for smartphones today. Based on data released by Statista, there are an estimated 2.2 billion android users worldwide by 2022 [1]. The large number of android users today makes the operating system a popular target for cyberattacks. Approximately 98.5% of smartphone attacks are primarily directed at Android devices [2]. One method of attack that is of much concern today is malware. Malware is defined as software that aims to infiltrate a computer system without the knowledge and consent of the system owner which can be intrusive, dangerous, and even destructive to the system [3]. Malware is one of the tools from attackers that internet users in the world must be aware of, including Android users.

Malware detection on Android is one of the important research topics in the field of cybersecurity today. One method that can be used to detect android malware is to manually analyze the malware, but this method is less relevant because it takes a lot of time [4]. Machine learning techniques are the right choice to deal with the complexity of android malware [5]. In general, methods that can be used in modeling android malware detection systems fall into three categories, namely supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning [6]. However, there is another development, namely ensemble learning, whose concept is to combine machine learning models. The application of ensemble learning provides improved performance of the detection model [7] [6].

The ensemble method is a type of machine learning algorithm that aims to improve prediction performance on a task by combining predictions from multiple estimators or models [8]. Bagging, Boosting, and Stacking are some of the popular and widely used ensemble techniques [9]. Stacking is the most popular meta-learning method and gets its name because it stacks a second classifier (meta learner) on top of its base classifier [8].

There are two methods that can be used for android malware detection, static analysis and dynamic analysis. The static method decompiles APK files and extracts features from files or program instructions, while the dynamic method runs the application in an isolated device [10]. Static features can be obtained quickly by parsing the APK file after the decompilation process. Static analysis has some limitations in the face of modifications from malware, but it is still more effective to use because it can quickly detect and prevent the installation process of malware [11].

The ubiquitous use of Android operating systems has made smartphones prime targets for cyber threats, particularly malware attacks. Defined as software infiltrating computer systems without the owner's consent, malware poses a grave risk to user privacy and system integrity. Existing manual analysis methods are time-consuming, underscoring the need for advanced approaches to Android malware detection. This study addresses this imperative by introducing an ensemble stacking framework, leveraging various classifiers to enhance the efficiency of Android malware detection. The research objectives involve a comprehensive evaluation of individual base classifiers, the implementation of regularization-based feature selection to optimize model accuracy, and a thorough comparative analysis to demonstrate the superiority of the proposed ensemble stacking framework over conventional Android malware detection methods. Through these specific objectives, the research aims to provide a novel and robust solution to the evolving cybersecurity threats faced by Android users.

In this paper, we introduce an ensemble stacking framework for Android malware detection. The contributions of this article can be summarized as below:

- A novel approach to Android malware detection was introduced, utilizing the combined power of ensemble stacking classifiers and regularization-based feature selection.
- Individual performances of base classifiers were extensively evaluated, providing foundational insights into their strengths and limitations.
- By employing Lasso, Ridge, and Elastic Net for feature selection from the Drebin-215 dataset, the ensemble model's accuracy was notably enhanced, with a concomitant reduction in computational overhead.
- A comprehensive comparative analysis was showcased, highlighting the distinct advantages of the proposed system over conventional Android malware detection methods.

The rest of the paper is summarized as follows in Section 2 provides a summary of relevant past research. In Section 3, the proposed methodology is laid out, encompassing the dataset description, the application of the regularization-based feature selection, and details of the ensemble stacking model. Section 4 presents the experiments and their results, beginning with the experimental setup, followed by the performance metrics, results from base classifiers, ensemble stacking outcomes, and a comparative analysis of the results. Finally, Section 5 concludes the paper and offers insights into potential avenues for future research.

## 2. RELATED WORKS

EnDroid is a dynamic analysis framework for accurate malware detection proposed in [12]. The research utilizes runtime behavior information obtained through the DroidBox tool, which captures various application activities and system calls. EnDroid uses feature selection to extract critical behavioral features, helping in the identification of malicious activities such as personal information theft and malicious service communication. The framework uses ensemble stacking method to achieve effective android malware detection. Overall, EnDroid's accuracy against all android malware samples reached 94.5% and the malware detected by EnDroid was almost evenly distributed across the malware families of the Drebin dataset.

Idrees et al. (2017) proposed PIndroid, an innovative android malware detection framework, which combines permissions and intents to identify malicious apps [13]. The framework uses a stepwise classification approach enhanced with ensemble techniques for optimized detection results. The proposed method achieved a remarkable accuracy of 99.8% when applied to 1,745 apps, making it one of the most accurate solutions reported to date. This study highlights the effectiveness of permissions and intents in detecting malware, and reveals their potential to detect malicious apps and other types of malware.

Yerima and Sezer (2018) proposed a fusion approach, called DroidFusion, to improve the predictive ability of machine learning algorithms in detecting android malware [14]. This framework combines a lower-level base classifier and applies a higher-level ranking-based algorithm to obtain the final classification. DroidFusion can utilize traditional machine learning algorithms (e.g., Decision Tree, Naive Bayes) and ensemble learning algorithms (e.g., Random Forest, Boosting) to improve classification accuracy. Extensive experiments were conducted on four datasets, including samples from Android Malgenome and DREBIN, to demonstrate the effectiveness of this approach. DroidFusion outperforms other stacking models, despite using only a high-level computational approach (meta classifier).

Rahman & Saha (2019) introduced StackDroid, a stacked generalization approach that combines base classifiers, including Extremely Randomized Tree (ET), Random Forest (RF), Multi-Layer Perceptron (MLP), and Stochastic Gradient Descent (SGD) in the first layer, with Extreme Gradient Boosting (XGBoost) as a meta classifier in the second layer. This approach achieved good results, with a detection accuracy of 97%, a False Positive Rate (FPR) of 1.67%, and an Area Under Curve (AUC) of 99% on the DREBIN dataset.

Dhalaria and Gondotra (2020) proposed an ensemble stacking method for android malware detection based on static and dynamic features [16]. The method involves using a chi-square feature selection algorithm to select features that are relevant in malware detection, thereby reducing the dimension of the feature vector. The selected features are used to build the base classifier, and the top three classifiers are combined with the ensemble stacking technique to improve the detection rate. The proposed approach achieves a high detection accuracy of 98.02% when evaluated on the AndroMD dataset, which outperforms existing classification engines. The experimental results demonstrate the effectiveness of the proposed approach in detecting Android malware.

Zhu et al. (2020) introduced SEDMDroid, an ensemble stacking framework that combines static analysis and machine learning [17]. SEDMDroid ensures individual accuracy and diversity in the ensemble by utilizing random feature subspaces and bootstrapping to generate diverse subsets. Principal component analysis (PCA) is applied to each subset. Experimental results on two datasets, including features such as permissions, sensitive APIs, and events from monitoring systems, demonstrate the effectiveness of SEDMDroid with accuracy rates of 89.07% and 94.92%. Comparison with existing methods further confirms the superiority of the proposed two-layer ensemble learning framework.

Wang et al. (2022) proposed an Android malware detection framework named MFDroid based on ensemble stacking, which overcomes the limitations of single feature selection and traditional machine learning methods [18]. The framework utilizes seven feature selection algorithms to process permissions, API calls, and opcodes features, which are then combined to form a new feature set. This feature set is then used to train five base classifiers in the first layer of the ensemble stacking framework, and utilize logistic regression as the meta classifier in the second layer. MFDroid achieved an F1 score of 96.0% for android malware detection.

Another method is GA-StackingMD introduced in [19]. GA-StackingMD combines five base classifiers using stacking, then optimized using genetic algorithm for hyperparameter tuning. This research uses a two-step feature selection method, utilizing information gain and chi-square. The proposed concept achieved a detection accuracy of 98.43% and 98.66% on the CIC-AndMal2017 and CICMalDroid2020 datasets, respectively.

Our proposed method introduces an ensemble stacking framework that adeptly integrates diverse base classifiers, including Random Forest, K-Nearest Neighbors, Support Vector Machine, Logistic Regression, and Bernoulli Naive Bayes, with Logistic Regression serving as the meta classifier. Applied to the real-world Drebin-215 dataset and enhanced with regularization-based feature selection techniques (Lasso, Ridge, and Elastic Net), our experiments underscore the robustness and efficacy of this approach in Android malware detection.

## 3. PROPOSED METHODOLOGY

In this study, an experimental research design was employed for the analysis of Android malware detection using the Drebin-215 dataset. The variables under investigation encompass features and attributes crucial for understanding malware instances within the Android environment. Specifically, we examine how various characteristics influence the effectiveness of ensemble stacking classifiers in detecting and mitigating malware threats. The proposed analysis criteria involve the utilization of ensemble stacking classifiers and regularization-based feature selection techniques. These methods were deliberately chosen to enhance the precision of our Android malware detection model. By employing ensemble techniques and feature selection, we aim to provide a robust and efficient approach to address the complex challenges posed by evolving malware threats in the Android ecosystem. The chosen research design and analysis criteria are intricately linked to our research objectives, ensuring a comprehensive exploration of the Android malware detection landscape and facilitating meaningful interpretation of the achieved results.

### 3.1 Dataset Description

The primary dataset underpinning this research is Drebin-215, a creation of Yerima and Sezer in 2019. This dataset was meticulously crafted with the specific intent of pioneering advancements in the domain of Android malware detection. Spanning across a diverse range of 15,036 APK files, the Drebin-215 dataset boasts 215 uniquely identified features. Delving deeper into its composition, it's noteworthy to highlight that of these APK files, 5,560 have been identified as malicious, while the remaining 9,476 are benign in nature. Each feature within this rich dataset was extracted through an exhaustive static analysis procedure applied to every individual APK file. To streamline and standardize representation, these features are encoded in binary format and distinctly categorized under 'malware' or 'benign' labels, ensuring clarity and precision.

For analytical and structural purposes, the dataset further organized into four primary classifications: API call signatures, manifest permissions, intents, and command signatures. The API call signatures are particularly fascinating as they represent the specific invocations of functions found within .dex file extensions. In contrast, the manifest permissions offer insights into the various permissions an application seeks, as explicitly declared within its inherent manifest file. Intents, another pivotal category, shed light on the intricate communication protocols established between different apps or services, as meticulously documented within the AndroidManifest.xml. Lastly, the command signatures serve as a repository of Linux commands, extracted from an array of file types including but not limited to .dex, .jar, .so, and .exe extensions.

### 3.2 Feature Selection using Regularization

In shaping predictive models for the Drebin-215 dataset, which is characterized by its vast dimensionality, the implementation of regularization techniques is crucial. These methods not only bolster the model's precision but also fortify its resilience against overfitting—a common challenge in high-dimensional spaces. The Lasso method applies L1 regularization to prune the feature set, diminishing certain coefficients to zero and thus performing dual roles in optimization and feature selection. Ridge regression, through L2 regularization, minimizes the coefficients' squares, ensuring that each feature's impact, while reduced, remains in the model. Elastic Net, adeptly combining L1 and L2 regularization, excels in situations where features are interdependent or excessively numerous.

Our choice to harness these regularization methods for the Drebin-215 dataset stems from their capacity to condense the feature space effectively. Lasso's sparsity-inducing approach is instrumental in discarding non-contributory features, streamlining the model in the process. Ridge is incorporated for its proficiency in managing feature interrelations without excluding any, thus preserving the integrity of the model's predictive power. Elastic Net is poised to capitalize on the combined strengths of Lasso and Ridge, offsetting their individual limitations and ensuring a balanced feature selection mechanism. This strategic selection is predicated on the belief that marrying the concepts of sparsity and controlled coefficient reduction can cultivate a model that offers both interpretability and wide applicability.

In our research, the feature selection is strategically designed to leverage the strengths of regularization techniques, crucial for sifting through the complex feature space of the Drebin-215 dataset. We propose using Lasso, Ridge, and Elastic Net methods, which are renowned for their efficacy in reducing overfitting and enhancing model interpretability. These techniques will be applied conceptually to identify and retain the most predictive features while discarding the redundant ones. The tuning of the regularization parameter alpha will be a focal point across all methods to determine the most effective level of penalization. For Elastic Net, the l1_ratio is preset at 0.5 to maintain a balanced approach between L1 and L2 regularization. This targeted calibration aims to construct a streamlined feature set, laying the foundation for a robust ensemble stacking classifier that is anticipated to improve the detection rates of Android malware significantly.

### 3.3 Ensemble Stacking Model

Ensemble stacking is a sophisticated machine learning technique that leverages the strengths of multiple classifiers to achieve superior predictive performance. Unlike traditional ensemble methods that often involve simple voting or averaging of predictions, stacking involves multiple layers of classifiers. The primary idea behind stacking is to use predictions from several base classifiers as input features for a higher-level, or meta, classifier. This hierarchical structure allows the model to learn from the diverse strengths of the base classifiers, potentially leading to enhanced accuracy and robustness. In our model, we utilize Random Forest, K-Nearest Neighbors, Support Vector Machine, Logistic Regression, and Bernoulli Naive Bayes as our base classifiers, with Logistic Regression serving as the meta classifier.

Before diving into the specifics of the chosen classifiers, it's worth noting that each classifier's performance can be heavily influenced by its hyperparameters. In our research, we recognized the importance of this and dedicated a substantial amount of time to rigorously tuning the hyperparameters for each classifier. This was done using random search techniques, ensuring that we harness the maximum potential of each base classifier in our ensemble stacking model.

#### 3.3.1 Base Classifiers

In our research, we opted for a combination of Random Forest, K-Nearest Neighbors, Support Vector Machine, Logistic Regression, and Bernoulli Naive Bayes as our base classifiers. The selection of these classifiers was driven by their diverse underlying methodologies, ensuring a broad spectrum of decision boundaries and prediction strategies. This diversity is crucial in ensemble methods, as it increases the chances of capturing various aspects and nuances of the data.

**Random Forest**: Random Forest was chosen for its ensemble nature, which inherently promotes diversity in decision-making. By constructing multiple decision trees and aggregating their outputs, RF offers a robustness against overfitting. Its ability to handle large datasets with higher dimensionality, manage missing values, and provide feature importance scores made it indispensable for our ensemble.

**K-Nearest Neighbors**: The inclusion of KNN was driven by its ability to capture local patterns and non-linear relationships in the data. As a non-parametric algorithm, KNN doesn't make strong assumptions about the underlying data distribution, making it versatile and adaptive to various data structures.

**Support Vector Machine**: SVM was selected for its efficacy in high-dimensional spaces and its capacity to find the optimal hyperplane using the kernel trick. This allows SVM to capture complex relationships in the data, even when they are not linearly separable. Its resilience against overfitting, especially with the right choice of kernel and regularization, made it a valuable addition to our ensemble.

**Logistic Regression**: Logistic Regression's probabilistic nature offers a clear advantage in scenarios where understanding the likelihood of class membership is crucial. Its ability to provide calibrated probabilities and its interpretability, where each feature's effect on the outcome can be quantified, made it a strategic choice for our ensemble, especially in the context of malware detection where understanding feature influence can be pivotal.

**Bernoulli Naive Bayes**: The decision to include BernoulliNB stemmed from its efficiency with binary/boolean features and its probabilistic foundation. Given that many features in malware detection can be binary (e.g., presence or absence of a certain permission), BernoulliNB's ability to handle such features efficiently and provide probabilistic outputs made it a complementary choice to the other classifiers.

#### 3.3.2 Meta Classifier

For the meta classifier, we selected Logistic Regression. The choice was motivated by several factors. Firstly, Logistic Regression's probabilistic nature is inherently suited for the role of a meta classifier. In ensemble stacking, the meta classifier is presented with predictions from multiple base classifiers, each with its own confidence or probability score. Logistic Regression's ability to weigh these probabilities in a nuanced manner ensures that the final decision is a balanced reflection of the collective wisdom of the base classifiers. This is especially crucial in malware detection, where the stakes are high, and a false positive or false negative can have significant implications.

Furthermore, the interpretability of Logistic Regression is a significant advantage in the context of our research. In the realm of Android malware detection, understanding the influence of each base classifier on the final decision can provide valuable insights into the nature of the malware and the features that are most indicative of malicious behavior. Logistic Regression, with its coefficients and odds ratios, offers a transparent view into this decision-making process, allowing researchers and practitioners to gain a deeper understanding of the detection mechanism. In summary, the selection of Logistic Regression as the meta classifier was not only driven by its technical merits but also by its alignment with the goals and requirements of our research in Android malware detection.

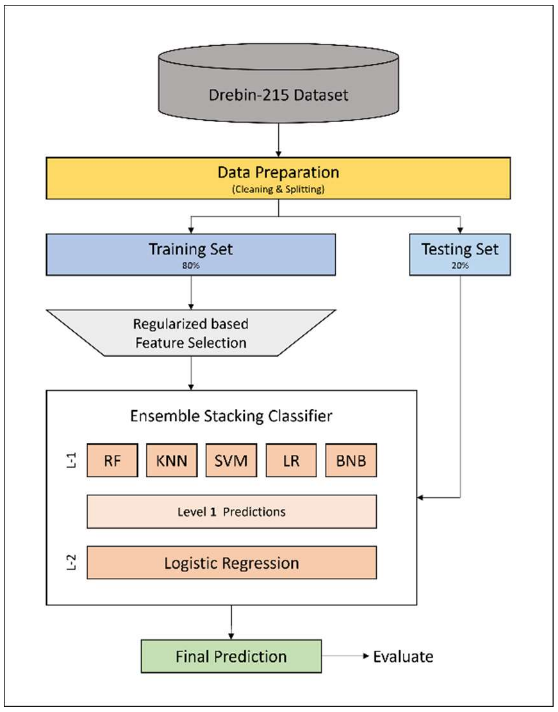

*Figure 1: Proposed Methodology*

## 4. EXPERIMENTS AND RESULTS

### 4.1 Experimental Setup

For the entirety of this research, a dedicated workstation was utilized, ensuring a consistent and controlled environment throughout the experimental phase. The workstation was equipped with an Intel(R) Core(TM) i7-7700HQ CPU, operating at approximately 2.8GHz with eight processing units. Accompanying this was a sizable 32GB DDR4 RAM operating at 2400MHz and an NVIDIA GeForce GTX 1070 graphics card. Storage capabilities were dual-faceted, encompassing a 1TB HDD with a spin rate of 7200rpm, complemented by a faster 512GB NVMe SSD, ensuring swift data read and write operations.

The primary toolkit for this venture was the scikit-learn library, a comprehensive machine learning tool in the Python ecosystem. It facilitated most of the experimental tasks, from model definition and training to evaluation. Furthermore, the nuances of feature selection using genetic algorithms were adeptly handled using the scikit-genetic-opt library, enhancing the optimization phase of the research.

Train-test split approach was adopted in this research, allocating 80% of the data for training and reserving 20% for testing. This method ensures that our model evaluations are based on a clear separation of data, with the training set used to develop the model and the testing set used to assess its performance. Such a split not only simplifies the evaluation process but also provides a focused measure of how well the model can generalize to new, unseen data, thereby offering a realistic perspective on its predictive capabilities.

In the quest to optimize our machine learning classifiers, we employed the RandomizedSearchCV methodology from scikit-learn. This approach allows for a more efficient and strategic exploration of the comprehensive hyperparameter grid we established for each classifier. Instead of exhaustively searching through every possible combination, RandomizedSearchCV samples a subset, striking a balance between thoroughness and computational efficiency. The detailed results of this hyperparameter tuning process, including the best parameters identified for each model, are meticulously presented in Table 1.

*Table 1: Hyperparameters of Classifiers*

| Classifier | Hyperparameters |
|---|---|
| Random Forest | n estimators = 200<br>max depth = none<br>criterion = gini<br>min samples split = 2<br>min samples leaf = 1<br>max features = log2 |
| K Nearest Neighbors | n neighbors : 1 |
| Support Vector Machine | C = 100<br>kernel = rbf |

| | |
|---|---|
| Logistic Regression | solver = liblinear<br>penalty = l1<br>C = 10 |
| Bernoulli Naïve Bayes | alpha = 0.001 |

## 4.2 Performance Metrics

In order to comprehensively assess the performance of the proposed android malware detection ensemble stacking approach, we utilize a set of well-established evaluation metrics. These metrics provide insights into various aspects of the detection model's effectiveness, covering its ability to accurately distinguish between benign and malicious applications. In this study, several performance metrics were used, including Accuracy, Error Rate, Precision, and Recall.

Central to our assessment is Accuracy, which measures the model's general predictive prowess by gauging the overall correctness of the classifier's predictions. Representing the proportion of correctly classified samples to the entire dataset, a heightened accuracy suggests a commendable predictive ability. Alongside, the True Positive Rate (TPR) or Sensitivity offers a deeper dive, evaluating the model's proficiency in identifying genuine threats—the malicious applications—amidst all actual positive instances. Complementing the TPR is Precision, which quantifies the exactness of the model's positive predictions and is vital for minimizing false alarms. Lastly, to offer a holistic view of the model's performance, the F-1 Score—a harmonious blend of precision and recall—provides a comprehensive assessment, especially crucial when dealing with skewed datasets. The formula of each metric for model evaluation used is shown in Table 2.

*Table 2: Evaluation Metrics*

| Evaluation Metric | Formula |
|---|---|
| Accuracy | $\frac{TP+TN}{TP+TN+FP+FN}$ |
| TPR | $\frac{TP}{TP+FP}$ |
| Precision | $\frac{TP}{TP+FP}$ |
| F1-score | $\frac{2*Precision*Recall}{Precision+Recall}$ |

## 4.3 Regularizatio-based Feature Refinement

Feature selection is a pivotal step in machine learning, especially for high-dimensional datasets like Drebin-215. In our study, we harness the power of regularization techniques to refine and select the most pertinent features for android malware detection. We employed three prominent regularization methods: Lasso, Ridge, and ElasticNet. Each of these techniques introduces a penalty to the model's loss function, shrinking some feature coefficients and thereby refining the feature set.

**Lasso (L1 regularization)**: Lasso penalizes the absolute values of the coefficients, leading to some feature coefficients becoming zero. This method resulted in retaining 194 features out of the original.

**Ridge (L2 regularization)**: Ridge penalizes the square of the coefficients, reducing their magnitude but not zeroing them. After applying Ridge, 77 features were retained.

**ElasticNet**: Combining the penalties of both L1 and L2 regularization, ElasticNet balanced the strengths of Lasso and Ridge. Post-application, 79 features were retained.

The reduced feature set post regularization ensures a more streamlined model, focusing on the most relevant indicators of malware presence. Table 3 provides a detailed breakdown of the number of features retained after applying each regularization technique. The selected features highlight the key characteristics crucial for distinguishing between benign and malicious applications.

## 4.4 Results of Base Classifiers

To discern the effectiveness of our ensemble stacking approach and the influence of feature refinement, we meticulously evaluated the base classifiers on different datasets: the original dataset, which encompasses all features, and three others refined through distinct feature selection techniques: Lasso (L-Drebin), Ridge (R-Drebin), and Elastic Net (EN-Drebin). This methodical, side-by-side evaluation is essential to understand the nuances of classifier performance and to shed light on the potential advantages or challenges introduced by feature selection. By comparing the results, we can elucidate the tangible impact of feature refinement on classifier accuracy, precision, recall, and overall effectiveness. Such a comparison not only validates the importance of an optimized feature set but also provides insights into how each classifier responds to changes in the input feature space. Detailed metrics, showcasing the depth of this evaluation for each classifier on all datasets, are systematically presented below.

In our ensemble exploration, the Random Forest (RF) classifier emerged as a notable contender. It adeptly navigated the intricate patterns of our dataset, demonstrating an initial performance with an accuracy of 98.64%, a recall of 96.88%, precision of 99.45%, and an F1 score of 98.15%. Upon the introduction of Lasso feature selection, its accuracy slightly improved to 98.74%, recall to 96.97%, precision to 99.63%, and F1 score to 98.28%. Ridge and ElasticNet feature selections, however, saw a decrease in performance with Ridge bringing the accuracy down to 97.57%, recall to 96.16%, precision to 97.29%, and F1 score to 96.72%, while ElasticNet resulted in an accuracy of 97.84%, recall of 96.34%, precision of 97.83%, and an F1 score of 97.08%. Figure 2 is illustrative of the comparative performance of the Random Forest classifier on each dataset, showcasing the impact of different feature selection methods on key metrics in a side-by-side manner.

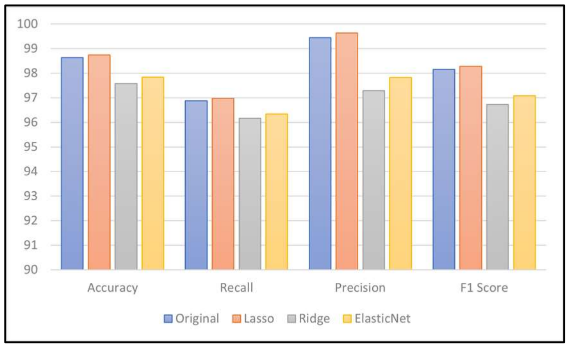

*Figure 2: Evaluation of Random Forest*

The K-Nearest Neighbors (KNN) classifier, with its proximity-based logic, initially exhibited an accuracy of 98.74%, a recall of 97.86%, precision of 98.74%, and an F1 score of 98.30%. The application of Lasso maintained the accuracy and F1 score at 98.74% and 98.30% respectively, while recall and precision saw slight changes to 98.13% and 98.48%. With Ridge, the accuracy declined to 97.57%, recall to 96.52%, precision to 96.95%, and F1 score to 96.74%. ElasticNet regularization achieved an accuracy of 97.81%, recall of 96.97%, precision of 97.14%, and F1 score of 97.05%. Figure 3 depicts the performance variations of the K-Nearest Neighbors (KNN) classifier, as evidenced by the changes in Accuracy, Recall, Precision, and F1 Score across the original, Lasso, Ridge, and ElasticNet feature-selected datasets.

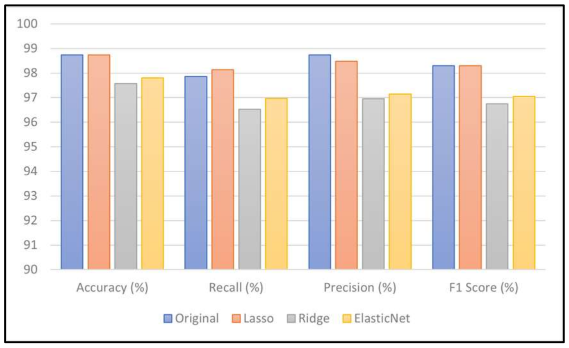

*Figure 3: Evaluation of K-Nearest Neighbors*

The Support Vector Machine (SVM) classifier began with a high accuracy of 98.94%, a recall of 98.39%, precision of 98.75%, and an F1 score of 98.57%. Post feature selection with Lasso, the metrics remained consistent, maintaining an accuracy and F1 score of 98.94% and 98.57% respectively, with a slight decrease in recall to 98.22% and an increase in precision to 98.92%. Ridge and ElasticNet feature selections resulted in a reduction of performance metrics, with Ridge leading to an accuracy of 97.41%, recall of 95.00%, precision of 97.98%, and F1 score of 96.47%, and ElasticNet yielding an accuracy of 97.81%, recall of 96.07%, precision of 98.00%, and F1 score of 97.03%. Figure 4 is indicative of the Support Vector Machine (SVM) classifier's performance metrics, detailing the comparative influence of the original and feature-selected datasets (Lasso, Ridge, ElasticNet) on Accuracy, Recall, Precision, and F1 Score.

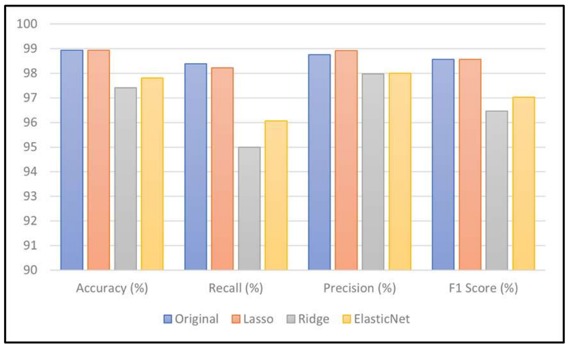

*Figure 4: Evaluation of Support Vector Machine*

The Logistic Regression (LR) classifier, grounded in probabilistic reasoning, initially recorded an accuracy of 97.74%, a recall of 97.15%, precision of 96.80%, and an F1 score of 96.97%. Lasso feature selection led to a marginal improvement in accuracy to 97.77%, with recall at 97.06%, precision at 96.97%, and F1 score at 97.01%. Ridge regularization saw a notable decrease in metrics, with accuracy at 96.41%, recall at 93.84%, precision at 96.43%, and F1 score at 95.12%. ElasticNet regularization resulted in an accuracy of 96.67%, recall of 94.56%, precision of 96.45%, and F1 score of 95.50%. Figure 5 presents the performance metrics of the Logistic Regression classifier, revealing the effects of the original and feature-selected datasets (Lasso, Ridge, ElasticNet) on Accuracy, Recall, Precision, and F1 Score in a comparative context.

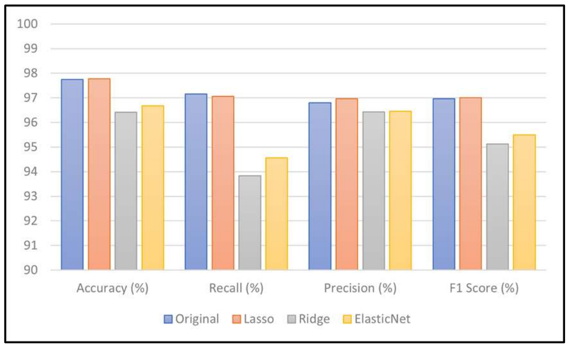

*Figure 5: Evaluation of Logistic Regression*

Lastly, the Bernoulli Naïve Bayes (BNB) model, which excels in handling binary data, began with an accuracy of 83.17%, a recall of 95.00%, precision of 70.30%, and an F1 score of 80.80%. The introduction of Lasso improved accuracy to 83.87%, recall to 95.63%, precision to 71.09%, and F1 score to 81.55%. Ridge regularization resulted in a significant improvement in metrics with accuracy at 88.83%, recall at 91.70%, precision at 80.88%, and F1 score at 85.95%. ElasticNet further enhanced the performance to an accuracy of 89.59%, recall of 93.40%, precision of 81.42%, and F1 score of 87.00%, emphasizing the adaptability and wide-ranging applicability of Bayesian classifiers with feature selection. Figure 6 displays the comparative impact on the Bernoulli Naive Bayes classifier's performance, with the datasets—original, Lasso, Ridge, and ElasticNet—each affecting the Accuracy, Recall, Precision, and F1 Score metrics.

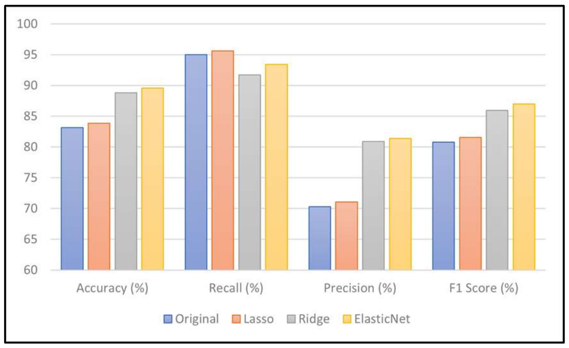

*Figure 6: Evaluation of Bernoulli Naive Bayes*

## 4.5 Ensemble Stacking Results

The core of our research hinges on the efficacy of the ensemble stacking approach. By integrating the strengths of multiple base classifiers, the stacking model aims to deliver superior performance, especially in the realm of Android malware detection. In this section, we delve into the performance of the ensemble stacking model, evaluated on different datasets: the original dataset, which encompasses all features, and three others refined through distinct feature selection techniques: Lasso (L-Drebin), Ridge (R-Drebin), and Elastic Net (EN-Drebin). The comprehensive performance metrics derived from these evaluations are detailed in Table 3.

*Table 3: Performance Metrics of Ensemble Stacking*

| Data | Accuracy | TPR | Precision | F-1 Score |
|---|---|---|---|---|
| Drebin-215 | 99.17 | 98.39 | 99.37 | 98.88 |
| L-Drebin | 99.24 | 98.31 | 99.64 | 98.97 |
| R-Drebin | 97.71 | 95.81 | 97.99 | 96.89 |
| EN-Drebin | 97.90 | 96.52 | 97.83 | 97.17 |

In our study, we compared the performance of an ensemble stacking classifier across four datasets, with updated metrics reflecting the latest findings. The Lasso feature selection method, applied to the L-Drebin dataset, now stands out even more with an enhanced accuracy of 99.24%. This improvement suggests that the Lasso's ability to identify a parsimonious yet effective feature set can indeed augment the classifier's discernment capability. The original Drebin dataset maintains a strong baseline with an accuracy of 99.17%, reinforcing the robustness of the initial feature set.

When considering recall, the original dataset exhibits a commendable rate of 98.39%, which is slightly higher than the L-Drebin's 98.31%, indicating that the initial features maintain a high sensitivity towards malware detection. The Ridge and Elastic Net feature selection methods, applied to the R-Drebin and EN-Drebin datasets, show a decrease in recall to 95.81% and 96.52% respectively. This decline may suggest that while these methods can help reduce feature dimensionality, they might also suppress some informative features essential for maintaining high sensitivity.

Precision is led by the L-Drebin dataset at 99.64%, a slight increase from the original dataset's precision of 99.37%. This further supports the Lasso method's efficacy in minimizing false positives and enhancing the classifier's exactness. The Ridge and Elastic Net datasets exhibit precision scores of 97.99% and 97.83%, respectively, which, while slightly lower, still demonstrate a high level of specificity in positive predictions.

The F1 Score, which balances precision and recall, is highest for the L-Drebin dataset at 98.97%, closely followed by the original dataset at 98.88%. The R-Drebin and EN-Drebin datasets show a more noticeable decrease in F1 Scores, recording 96.89% and 97.17% respectively. These figures reflect the nuanced impact of feature selection on the classifier's overall performance, highlighting the trade-offs between reducing feature space and maintaining high detection capability.

Lasso feature selection proved to be the most beneficial for the ensemble stacking classifier in our study, improving its performance across all metrics. The original set of features was already strong, but Lasso made it even better. The Ridge and Elastic Net feature selection methods did not help as much and in some cases slightly lowered the classifier's effectiveness. This emphasizes the importance of choosing the right feature selection method to ensure the best performance of a malware detection classifier.

## 5. CONCLUSION

In this research, we presented an advanced method for Android malware detection by employing ensemble stacking classifiers coupled with regularization-based feature selection on the Drebin-215 dataset. This method was designed with a clear focus on addressing the initial problem statement and achieving the predefined research objectives. Our ensemble stacking approach effectively combined the capabilities of diverse base classifiers: Random Forest, K-Nearest Neighbors, Support Vector Machine, Logistic Regression, and Bernoulli Naive Bayes, with Logistic Regression acting as the meta classifier. The introduction of Lasso, Ridge, and Elastic Net was pivotal in optimizing the feature selection process, emphasizing the ascendancy of advanced regularization techniques in the realm of malware detection. Following this, the derived results were noteworthy, manifesting remarkable accuracy and an admirable F1-score. Furthermore, our model consistently exhibited elevated values for TPR and precision, highlighting its capability to accurately identify genuine malware instances while curtailing false positives. While there are challenges, such as potential variability among base classifiers and the constraints of static features, our method marks a significant stride in the relentless pursuit of shielding the mobile domain from malware adversities.

This study, while contributing significantly to Android malware detection, acknowledges inherent limitations and assumptions. Relying on the Drebin-215 dataset, comprehensive yet potentially limited in capturing real-world diversity, poses a constraint. The experimental nature of our design may restrict generalizability across contexts. Assumptions about Android malware behavior introduce variability in result applicability. Unattended issues surfaced, including the impact of evolving malware tactics and the dynamic Android ecosystem. Beyond this study's scope, these issues merit attention in future research. This transparent discussion aims to offer a nuanced understanding of findings' scope and applicability.

## REFERENCES:

[1] P. Taylor, "Global number of internet users 2012-2022, by operating system," 2023.

[2] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention," IEEE Transactions on Dependable & Secure Computing, vol. PP, no. 99, pp. 1-1, 2018.

[3] T. Alsmadi and N. Alqudah, "A Survey on malware detection techniques," in 2021 International Conference on Information Technology (ICIT), 2021, pp. 371–376.

[4] C. A. B. de Andrade, C. G. de Mello, and J. C. Duarte, "Malware Automatic Analysis," in 2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence, 2013.

[5] O. C. Abikoye, B. A. Gyunka, and O. N. Akande, "Android malware detection through machine learning techniques: A review," International Journal of Online and Biomedical Engineering, vol. 16, no. 2, pp. 14–30, 2020.

[6] K. Liu, S. Xu, G. Xu, M. Zhang, D. Sun, and H. Liu, "A Review of Android Malware Detection Approaches Based on Machine Learning," IEEE Access, vol. 8, pp. 124579–124607, 2020.

[7] M. Ozdemir and I. Sogukpinar, "An Android Malware Detection Architecture based on Ensemble Learning," Transactions on Machine Learning and Artificial Intelligence, vol. 2, no. 3, pp. 90–106, 2014.

[8] G. Kunapuli, "Ensemble Methods for Machine Learning," 2023.

[9] R. Odegua, "An empirical study of ensemble techniques (bagging, boosting and stacking)," in Proc. Conf.: Deep Learn. IndabaXAt, 2019.

[10] N. Xie, Z. Qin, and X. Di, "GA-StackingMD: Android Malware Detection Method Based on Genetic Algorithm Optimized Stacking," Applied Sciences (Switzerland), vol. 13, no. 4, 2023.

[11] Y. Pan, X. Ge, C. Fang, and Y. Fan, "A Systematic Literature Review of Android Malware Detection Using Static Analysis," IEEE Access, vol. 8, pp. 116363–116379, 2020.

[12] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, "A novel dynamic android malware detection system with ensemble learning," IEEE Access, vol. 6, pp. 30996–31011, 2018.

[13] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, and Y. Rahulamathavan, "PIndroid: A novel Android malware detection system using ensemble learning methods," Computers & Security, vol. 68, pp. 36–46, 2017.

[14] S. Y. Yerima and S. Sezer, "DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection," IEEE Transactions on Cybernetics, vol. 49, no. 2, pp. 453–466, 2018.

[15] S. S. M. Rahman and S. Saha, "StackDroid: Evaluation of a Multi-level Approach for Detecting the Malware on Android Using Stacked Generalization," in the book title (if available), ISBN: 978-981-13-9180-4, July 2019, pp. 611-623. doi:10.1007/978-981-13-9181-1_53

[16] M. Dhalaria and E. Gandotra, "Android malware detection using chi-square feature selection and ensemble learning method," in PDGC 2020 - 2020 6th International Conference on Parallel, Distributed and Grid Computing, 2020, pp. 36–41.

[17] H. Zhu, Y. Li, R. Li, J. Li, Z. You, and H. Song, "SEDMDroid: An Enhanced Stacking Ensemble Framework for Android Malware Detection," IEEE Transactions on Network Science and Engineering, vol. 8, no. 2, pp. 984–994, 2021.

[18] X. Wang, L. Zhang, K. Zhao, X. Ding, and M. Yu, "MFDroid: A Stacking Ensemble Learning Framework for Android Malware Detection," Sensors, vol. 22, no. 7, 2022.

[19] N. Xie, Z. Qin, and X. Di, "GA-StackingMD: Android Malware Detection Method Based on Genetic Algorithm Optimized Stacking," Applied Sciences (Switzerland), vol. 13, no. 4, 2023.