MALWARE CLASSIFICATION USING EXTRA TREE FEATURE SELECTION AND OVERSAMPLED ENSEMBLE MACHINE LEARNING

Al Fatehah Fahmi Abdurrahim Al Gozali1,a), Denar Regata Akbi2,b), Vinna Rahmayanti Setyaning Nastiti 3,c)

Author Affiliations

1,2,3Universitas Muhammadiyah Malang   
  
Author Emails

a) aagfahmi08@gmail.com,b)Corresponding author: dnarregata@umm.ac.id,

c)vinastiti@umm.ac.id

**Abstract.** *Malware* is one the most severe cyber threats, targeting the confidentiality, integrity, resources, and availability of data and systems without user consent. Its detection is challenging due to the use of advanced encryption techniques, making data recovery difficult. This study investigates *malware* classification using an ensemble machine learning approach with six base learners: *Random Forest, Logistic Regression, SVM, KNN, Multi-Layer Perceptron,* and *Decision Tree.* the models are combined through a stacking technique, with an averaging method applied o reduce variance and improve stability. Performance is evaluated on the **CCCS-CIC-AndMal2020** dataset, consisting of 53,750 *Android* application samples categorized into 14 *malware* classes and described by 144 features. Extra-tree feature selection is applied in two scenarios: with and without outlier handling, reducing feature from 144 to 125 and 55, respectively. Result show that *Random* *Forest* achieved the highest stand alone with 88% accuracy. Oversampling using SMOTE significantly improved the stacking model’s performance across all metrics, with the R2 score increasing from 0,7063 to 0,9329. Five-fold cross validation demonstrated consistent accuracy between 0,875 and 0,890. The model achieved high classification performance for *Riskware, Trojan Banker,* and *Trojan Dropper* with accuracy 0,926. The result obtained from this study show that the model has an *accuracy* rate of 97,11%, 93,29% for *R2*, *F1-Score* has 97,10%, 97,11% for *recall,* and *Precision* has 97,12%. Overall, the findings confirm that combining stacking with SMOTE effectively addresses class imbalance and enhances *malware* detection performance.

## INTRODUCTION

Malware is one of the most dangeroustypes of software. It targets the confidentiality, integrity, resources, or availability of data and systems without the victim’s consent [1]. This type of attack is considered one of the most severe classes of cybercrimes due to its high level of difficulty to detect. Malware often employs advanced dataencryption techniques, making data recovery extremely difficult once a system is infected.Based on its classification, malware can be divided into several types according to its behavior and characteristics, such as: *adware*, *worms*, *viruses*, *rootkits*, *trojan horses*, *backdoors*, *spyware*, *logic bombs*, and *ransomware* [2]. It can be used to steal personal data and information or exploit the victim’s computing resources to launch attacks on other targets. On the other hand, the formation of *anti-malware* software has also been widely discussed in several asticles and journals. The methods of analyzing and detecting *malware* themselves have many approaches, such as: environmental analysis (network, personal computer, mobile devices, cloud systems), data analysis (honeypot, static analysis, dynamic analysis, statistics), machine learning (*Bayesian,* decision tree, dimension reduction, instance based, clustering, deep learning, ensemble, neural network, regularization, rule system, regression) [3]. Several studies have discussed analytical methods for handling *malware*, which are implementations of various machine learning techniques.

The first study was conducted by Achmad Rizal Yogasware, Denar Regata Akbi, and Vinna Rahmayanti Setyaning Nastiti [1]. This research classifies *malware* families using the *K-Nearest Neighbors* method on secondary data from *Canadian Institute for Cybersecurity* in 2019. Using a 90:10 train-test split, KNN achieved 65% recall and 83% precision. Preprocessing with the C5.0 algorithm was noted to enhances classification performance.

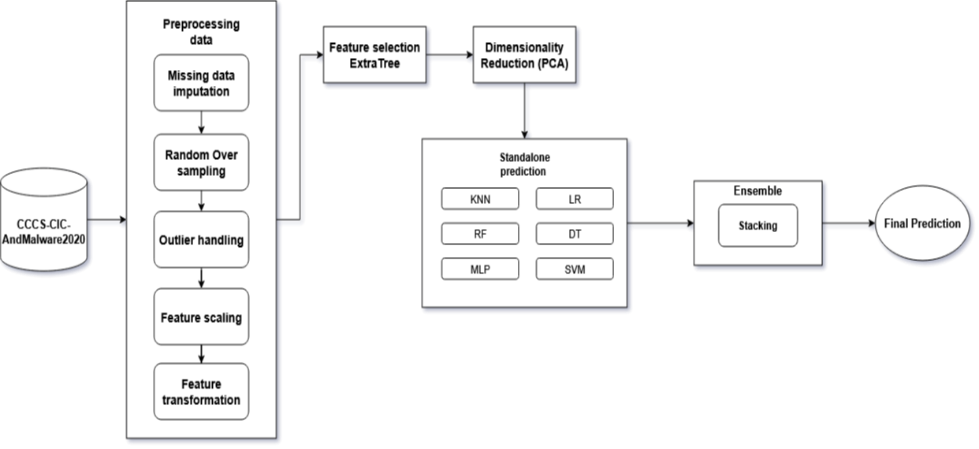
The second study was conducted by Ahmed Hashem El Fiky, Mohamed Ashraf Madkour, and Ayman El Shenawy [4]. This study presents a parallel machine learning model for dynamic detection of *Andorid* *malware* categories and families using the **CCCS-CIC-AndMal2020** dataset (14 categories, 180 families). The model achieved over 96% accuracy in category detection and over 99% in family identification, outperforming recent methods while significantly reducing acnalysis time on smartphones.

The third study was conducted by Vishnu Sripriya Akondi, Vineetha Menon Jerome Baudry, and Jana Whittle [5]. This research discusses the performance of feature selection in improving the accuracy of the *K-means* clustering method. The result of study indicated that the most effective feature selection method to enhance the accuracy of *K-means* clustering is the *Extra-tree* feature selection approach.

The forth study wa conducted by Rejwana Islam, Moinul Islam Sayed, Sajal Saha, Mohammad Jamal Hossain, and Md Abdul Masud [2]. this study employs a weighted voting ensemble of six classifier on the **CCCS-CIC-AndMal2020** dataset for dynamic multi-classification of *Andorid* applications and *malware*. using R2 base weighting, the model achieved 95% accuracy despite reducing features by 60,2%.

The fifth study was conducted by Dwinanda Bagoes Ansori, Joko Slamet, Muhammad Zakky Ghuffron, Muhammad Aidiel Rachman Putra, Tohari Ahmad [6]. This study proposed an approach for detecting *Android malware* and classifying it into five categories by using *gain ratio* feature selection and an ensemble machine learning algorithm. Feature were reduced based on their importance values calculated through the *gain ratio* method. Subsequently, the most significat features were used in a classification process that combined multiple models. Experiments conducted using the **CIC-AndMal-2020** dataset demonstreated that the proposed approach improved detection performance. Result has increased detection accuracy acros several machine learning algorithms: by 2,59% for *Naïve Bayes,* 0,90% for *KNN,* and 2,29% for *SVM*.

## METHODOLOGY



**FIGURE 1.** Framework of our proposed methodology.

Essentially, the ensemble machine learning technique combines multiple models to enhance classification accuracy and robustness by leveraging the strengths of diverse models. This approarch reduces the risk of overfitting and improves predictive performance, especially when the individual model are uncorrelated. As illustrated in figure 1, six types of models are employed in this study: *Random Forest, Logistic Regression, Support Vector Machine, K-Nearest Neighbors, Multi-Layer Perceptron, and Decision Tree.* these models serve as base learners for generating predictions, which are then combined through a stacking approach into a single model. To reduce variance and improve model satbility, the regression outputs from al base models will be averaged using the *Averaging* method to produce the final prediction. To combined model will then be analyzed and its accuracy compared between two scenarios: one using *Extra-tree* feature selection and the other without it. The result of this analysis and comparison will form the basis for evaluating the performance and effectiveness of the porposed model.

### **Dataset**

**CCCS-CIC-AndMal-2020**is a publicly available dataset released in 2020 by the *Canadian Center for Cyber Security* and the *Canadian Institute for Cybersecurity* [7]*.* It is a large collection of *Andorid* applications totaling approximately 400,000, consisting of 200,000 beign application and 200,000 malicious applications [4],[8]. For dynamic analysis, the dataset contains 144 categorizable features. In addtion, it includes 53,750 samples classified into 14 distinct types of *malwares* [9]*,*[10]*.*

**TABLE 1.** Dataset Structure.

|  |  |  |  |
| --- | --- | --- | --- |
| Categories | Families | Samples | Feature |
| Adware before reboot | 42 | 5.143 | 144 |
| Adware after reboot | 41 | 5.838 | 144 |
| Backdoor after reboot | 10 | 546 | 144 |
| Backdoor before reboot | 10 | 591 | 144 |
| FileInfector after reboot | 5 | 119 | 144 |
| FileInfector before reboot | 5 | 129 | 144 |
| PUA after reboot | 9 | 625 | 144 |
| PUA before reboot | 9 | 665 | 144 |
| Ransomware after reboot | 8 | 1.861 | 144 |
| Ransomware before reboot | 8 | 1.861 | 144 |
| Riskware after reboot | 18 | 6.792 | 144 |
| Riskware before reboot | 19 | 7.261 | 144 |
| Scareware after reboot | 4 | 424 | 144 |
| Scareware before reboot | 4 | 462 | 144 |
| Trojan after reboot | 37 | 4.025 | 144 |
| Trojan before reboot | 38 | 4.412 | 144 |
| Trojan\_Banker after reboot | 10 | 123 | 144 |
| Trojan\_Banker before reboot | 11 | 118 | 144 |
| Trojan\_Dropper after reboot | 10 | 733 | 144 |
| Trojan\_Dropper before reboot | 9 | 837 | 144 |
| Trojan\_SMS after reboot | 10 | 911 | 144 |
| Trojan\_SMS before reboot | 10 | 1.028 | 144 |
| Trojan\_Spy after reboot | 10 | 1.039 | 144 |
| Trojan\_Spy before reboot | 11 | 1.801 | 144 |

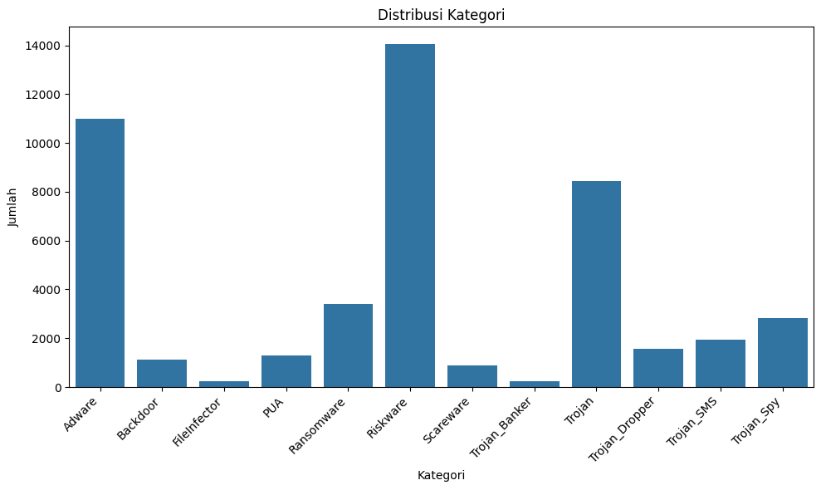
### **Preprocessing.**

First step preprocessing is exploration data analysis, the fisrt step involves exploring the dataset to understand its structure and content using function such as *head(), info(),* and *describe()*. Data cleaning, missing values (*null values*) are removed or imputed using methods such as mean, median, interpolation, or mode imputation. This ensures dataset completeness, prevent errors due to missing values, and minimizes data distortion. Outlier handling (optional), significant deveiation from general data patterns are detected and addressed using the *interquatile range* (IQR) methods to maintain the validity and accuracy of analysis. Data transformation, for normalization we applying *MinMaxScaler()* to rescale numeric values into a uniform range, while for encoding we using *LabelEncoder* to convert catagorical variables into numerical form for model preprocessing. Data splitting, the dataset is split into 60% training set (28,219 samples), 20% validation set (9,406 samples), and 20% test set (9,406 samples).

### **Missing Data Imputation**.

Many supervised machine learning methods cannot be applied to dataset containing missing values, such as regression technique or neural network [2]. To ensure the dataframes is free from missing values, the *SimpleImputer* from Scikit-Learn library was used with the *mean* strategy. The first step involved replacing all infinite values (both positive and negative) with *Numpy np.inf,* which represents values greater than any real number. Next, all empty values were converted to *Numpy np.nan* to indicate missing data, allowing algorithms to ignore them during computation. Finally, all *np.nan* and *np.inf* values were replaced with the mean value of the corresponding feature.

### **Random Oversampling**



**FIGURE 2.** Category Distribution.

Figure 2 illustrates and visualizes the distribution of each *malware* category in the **CCCS-CIC-AndMal2020** dataset. According to the dataset structure in Table 1, the number of *Trojan\_banker* sampels is 141, while *Riskware* has 14,053 samples. This imbalance can affect the performance of machine learning algorithms during the training phase. The **CCCS-CIC-AndMal2020** dataset is considered imbalanced in term of sample counts, as show in Figure 2. The frequency ratio between *Trojan\_banker* and *Riskware* is 1:58. This poses a challenge because some algorithms tend to overlook minority classes, even though prediction for minority classes are often more important. To address this class imbalance, a resampling technique was applied. The method used is the *Synthetic Minority Oversampling Technique* (SMOTE), with the simplest approach being duplicating samples from the minority class.

### **Outlier Handling.**

A data point, number, or value that deviates significantly from the majority of the data is called an *outlier.* The presence of outliers can affect the mean, median, standart deviation, and interpretation of analyis result, which many ultimately cause a model or algorithm to become inaccurate or biased. Moreover, outliers can obscure the true patterns in the data, resulting in bias. To prevent this, outlier handling is necessary before feeding data into a model. The **CCCS-CIC-AndMal2020** dataset has highly *skewed* distribution, causing the distribution tail to be longer in one direction. This skeweness is due to the characteristics of the data collected for dynamic *Android malware* analysis. Each *malware* type has different operational characteristics, which typically lead to varying values related to memory management, logcat, API calls, and so on. The technique used to identify outliers is the *Interquartile Range (IQR)*, which is calculated using the foolowing formula:

A black text on a white background

AI-generated content may be incorrect.

Where is interquartile range, is first quartile, is third quartile, and is recent values.

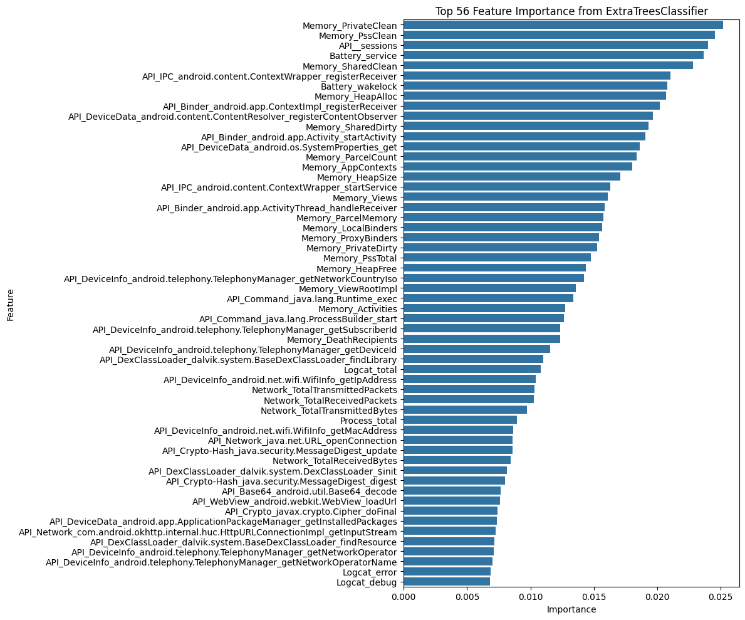
### **Feature Scaling.**

If the values of feature in a machine learning algorithm are similar to each other, there is agreater likelihood that the algorithm will be trained more efectively and quickly compared to a dataset with highly varied feature values. Dataset with widely differing feature values generally require more time for the model to learn and tend to produce lower accuracy. The function of feature scaling is to normalize or equalize the range of values across the dataset’s features, ensuring that all features are on the same or uniform scale. This is important because features in dataset often have very different value ranges. Without scaling, features with larger valuesmay dominate the model training process, which can result in biased and less accurate predictions. In this study, the researcher employed the *MinmaxScaler* from the *Scikit-Learn* library to normalize the data for each feature based on the minimum and maximum values within that feature.

### **Categorical Values Encoding.**

Since the **CCCS-CIC-AndMal2020** dataset is categorical, it requires transformation into intergers. The resercher utilized the *One-Hot-Encoder* form the *Keras* library to convert categorical data in interger form into binary vector representations, where each category is represented by a vector containing a single element with values 1 and rest with the values 0. This means that a feature represented by a given column will have a value of 1 if it belongs to the category converted into a binary feature, and 0 otherwise.

### Feature Selection.



**FIGURE 3.** Feature Selection using ExtraTreeClassifier.

Since the **CCCS-CIC-AndMal2020** is categorical with large number of features, it is necessary to apply a method for optimally selecting the most influential feature using a feature selection technique. This technique aims to identify the most relevant and important subset of features from the entire dataset in order to improve the performance of the machine learning model. In this study, the *Extra-tree* feature selection method we employed, considering the complex and categorical nature of the **CCCS-CIC-AndMal2020** dataset. In general, *Extra-tree* operates in a manner similar to decision tree. Once random splits are generated, *importance* scores are automatically calculated to indicated the relative contribution of each feature to the model predictions. Features with high *importance* scores are selected for calssification, while features with low *importance* score are discarded to reduce model complexity, thereby minizing the risk of overfitting. In the experiments, without outlier handling, 125 feature were found to be related to the classification labels out of the total 144 features. With outlier handling applied 56 features were found to be related to the classification, as explained in the Figure 3.

### **Dimensionality Reduction.**

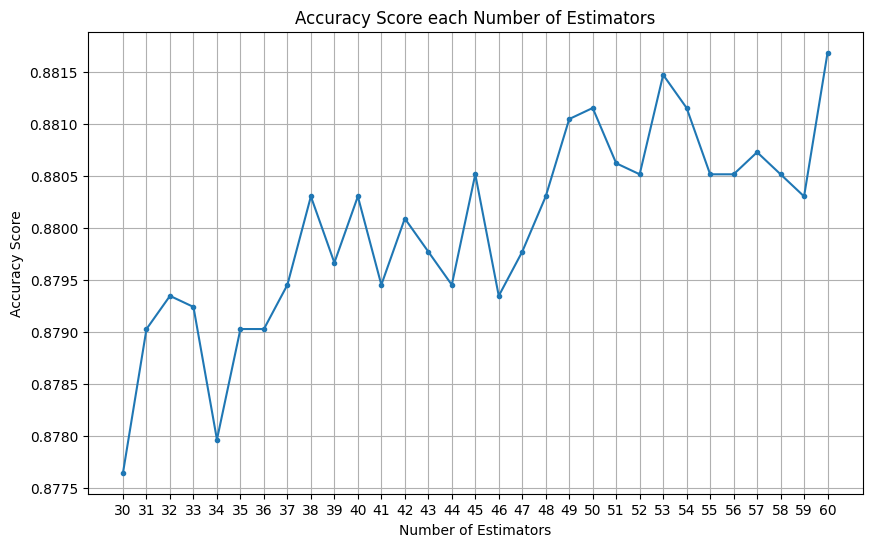
In this study, the reserachers applied *Principal Component Analysis (PCA)* and *t-distributed Stichastic Neighbor Embedding (t-SNE)* to the dataset. PCA tranforms data from a high-dimensional space into a lower-dimensional space, aiming to keep the lower-dimensional representation as close as possible to the original data. In analyses involving large numbers of observatons or variables, dimensionality reduction is highly beneficial, as smaller dataset are easier to analyze and visualize.

|  |  |
| --- | --- |
|  |  |
| (a) | (b) |

**FIGURE 4.** Dimensionality Reduction using PCA and t-SNE.

Figure 4 (a) present a PCA *biplot* after applying the *Outlier Capping* process to each category. The PC1 (Principal Component) axis represents the largest vaiability within the dataset and serves to separate various *malware* categories, while the PC2 axis capture significat variability that helps distinguish between closely related categories. It can be observed that certain categories are more concentrated in specific areas, particularly in the upper region of the plot, indicating clear clustering for categories with high density. Furthermore, there ara high-density regions that indicate freaquent occurences of certain categories. Figure 4 (b) present a *biplot* generated by t-SNE for dimensionality reduction after applying the *outlier capping* process. This *biplot* is useful for identifying distribution patterns of the data in its reduced- dimensional space. Clear clustering between categories can be observerd, indicating that the data is well-organized based on *malware* types. Some categories show high-density regions, such as *Trojan*, which appears widely distributed, and *Ransomware*, which exhibits a certain degree of concentration. Meanwhile, other categories appear more scattered. Futher analysis can be conducted to explore the specific characteristics of each cluster.

### **Hyperparameter Tuning.**



**FIGURE 5.** Accuracy Model RF for each n\_estimators.

The implementation of *GridSearchCV,* purpose of *hyperparameter tuning* is to search for and determine the optimal combination of *hyperparameter* values so that a machine learning model can perform at its best on a given dataset. This process allows the model to adapt to the characteristic of the data, improve accuracy, reduce the risk of *overfitting* or *underfitting*, and enhance overal prediction results. To evaluate the *hyperparameters*, the reserchers used *GridSearchCV* to identify the best combination of *hyperparameters* for machine learning model. This tool tests various parameters combinations to determine the best performance based on predefined metrics. Ther process inherently applies a five-fold *cross-validations*, which makes the model evaluation more accurate*.* For RF models, there are *hyperparameters tuning* such as; n\_estimator = 60, for max\_depth = ‘None’, min\_samples\_split = 2, and random\_state= 42, with achieving accuracy 0,8826 as can be seen in the Figure 5. For KNN model we used n\_neighbors = 3, with achieving accuracy 0,7470. MLP model, there are *hyperparameters tuning* such as; random\_state = 3, max\_iter = 300, activation= ‘relu’, alpha = 0,0001, hidden\_layer\_sizes = 100, with achieving accuracy 0,8010. For DT we used; random\_state = 36, max\_depth = 20, min\_samples\_split = 2, with achieving accuracy 0,8175. For SVM, there are *hyperparameter tuning* such as; Kernel = ‘rbf’, gamma = 10, C = 100, with achieving accuracy 0,8155. Figure 5 shows the *Random Forest* model’s accuracy for varying *n\_estimators*. Accuracy fluctuates between 30 and 60 estimators but generally improves with more estimators, peaking at about 0,8815 at 60 *n\_estimators.* Some variations suggest other factor may also affect accuracy.

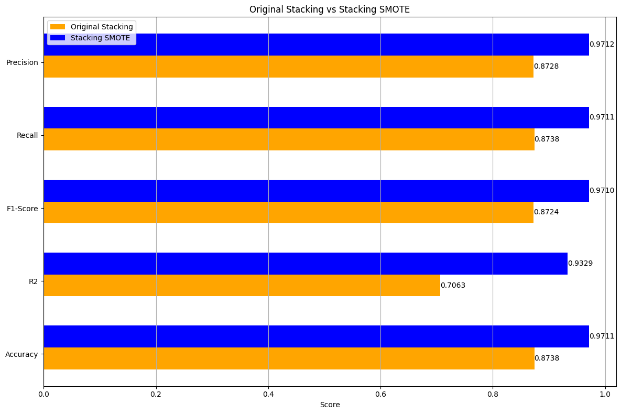
## RESULT AND DISCUSSION

### **Comparison Model**

|  |  |
| --- | --- |
|  |  |
| (a) | (b) |

**Figure 6.** Comparison Model when standing alone.

Figure 6 (a) present the accuracy comparison of each model without applying random oversampling, while Figure 6 (b) shows the result after the process was applied. The *Logistic Regression* model achieved an accuracy of 51%, a decrease of 8% from its previous score. The *K-Nearest Neighbors* model experienced a 2% drop, from 75% to 73%. The *Multi-Layer Perceptron* model decreased by 4%, from 80% to 76%. The *Decision Tree* model recorded an accuracy of 79%, down 2% from 81%. The *Support Vector Machine* model saw a slight decline of 1%, from 82% to 81%. Meanwhile, the *Random Forest* model maintained a consistent accuracy of 88%, remaining the highest among the six models. It appears that the oversampling process using SMOTE has a significant effect on the model learning process when standing alone.



**Figure 7.** Performance Ensemble Machine Learning.

### **Oversampled Ensemble Machine Learning**

In Figure 7, it is evident that ensemble learning using stacking with oversampled data demonstrates better performance across all matrics compared to stacking without oeversampling. The most significant improvement is observed in the R2 score, where stacking with SMOTE achieved 0,9329, while the original stacking only reached 0,7063. Performance in other metrics, such as precision, recall, F1-score, and accuracy, also indicates that stacking with SMOTE is more effective for classification. This suggests that applying the SMOTE technique in stacking can enhance the model’s ability to address class imbalance issues.

### **Evaluation Model**

|  |  |
| --- | --- |
|  |  |
| (a) | (b) |

**Figure 8.** Accuracy of Five-Fold Cross Validation.

Figure 8 (a) present the accuracy levels from five-fold cross validation. The first fold achieved an accuracy of 0,875, the second fold 0,890, the third fold 0,875, the fourth fold approximately 0,880, and the fifth fold around 0,890, this pattern shows that the second and fifth folds have the highest accuracy, indicating strong model performance in those folds. In contrast, the third fold shows the lowest accuracy, wich may be due to specific factor in the data used for that fold. Overal, the model demonstrates consistent performance with slight fluctuations, despite some variation across folds. The highest accuracies in certain folds indicates that the model’s efficiency in Classification is genrally quite good. Figure 8 (b) shows the confusion matrix of the ensemble machine learning model after oversampling. This matrix is used to evaluate the classification performance by comparing the actual labels with the predicted labels. As shown in Figure 8 (b), the model performs well for most classes, particularly for the *Riskware, Trojan\_Banker,* and *Trojan\_Dropper* categories, each achieving an accuracy of over 0,926. However, the *FileInfector* category shows a relatively low accuracy of around 0,275, which may indicate classification issues specific to that category.

# CONCLUSION

*Malware* is one of the most dangerous types od software, targeting the conficentiality, integrity, resources, or availability of data and systems without the victims’s consent. Such attacks are difficult to detect and often employ advanced encryption techniques, making data recovery challenging. *Malware* analysis and detection can be carried out through various approaches, such as environmental analysis, data analysis, and machine learning. this study employe six of these models as base learners, combined through stacking. The averaging method is applied to reduce variance and improve model stability. Performance is compared between models with and without *Extra-tree* feature selection. Without outlier handling, 125 out of 144 features are relevant; with outlier handling, only 56 features remain relevant. *Random Forest* achieved the highest accuracy (88%). Oversampling using SMOTE enhanced the stacking model’s performance across all metrics, especially R2 (0,9329). The five-fold Cross-Validation result show consistent accuracy between 0,875 and 0,890. The model performs exceptionally well for the *Riskware, Trojan\_Banker,* and *Trojan\_Dropper* categories, with achieved accuracy 0,926, but shows weaker performance for *FileInfector*. Although the accuracy of the model when stand alone with SMOTE is quite significant, it is possible that the same thing will happen during the ensemble learning process. The result obtained from this stady show that the model has an *accuracy* rate of 97,11%, 93,29% for *R2*, *F1-Score* has 97,10%, 97,11% for *recall,* and *Precision* has 97,12%. It has been proven that oversampling using SMOTE can improve the performance of ensemble’s stacking models.

# References

[1] A. R. Yogaswara, “Klasifikasi Malware Family menggunakan Metode k-Nearest Neighbor (k-NN),” *J. Repos.*, vol. 3, no. 3, pp. 305–314, 2021, doi: 10.22219/repositor.v2i3.1313.

[2] I. . Fernando, “Android Malware Classification Using Optimum Feature Selection and Ensemble Machine Learning,” *Tour. Covid-19 Pandemic a Perspect. With Swot Anal.*, vol. 15, no. 2, pp. 14–21, 2020.

[3] B. Yamany, M. S. Elsayed, A. D. Jurcut, N. Abdelbaki, and M. A. Azer, “A New Scheme for Ransomware Classification and Clustering Using Static Features,” *Electron.*, vol. 11, no. 20, Oct. 2022, doi: 10.3390/electronics11203307.

[4] A. H. El Fiky, M. A. Madkour, and A. El Shenawy, “Android Malware Category and Family Identification Using Parallel Machine Learning,” *J. Inf. Technol. Manag.*, vol. 14, no. 4, pp. 19–39, 2022, doi: 10.22059/jitm.2022.88133.

[5] V. S. Akondi, V. Menon, J. Baudry, and J. Whittle, “Novel K-Means Clustering-based Undersampling and Feature Selection for Drug Discovery Applications,” *Proc. - 2019 IEEE Int. Conf. Bioinforma. Biomed. BIBM 2019*, no. Mcl, pp. 2771–2778, 2019, doi: 10.1109/BIBM47256.2019.8983213.

[6] D. B. Ansori, J. Slamet, M. Z. Ghufron, M. A. R. Putra, and T. Ahmad, “Android Malware Classification Using Gain Ratio and Ensembled Machine Learning,” *Int. J. Saf. Secur. Eng.*, vol. 14, no. 1, pp. 259–266, 2024, doi: 10.18280/ijsse.140126.

[7] A. Rahali, A. H. Lashkari, G. Kaur, L. Taheri, F. Gagnon, and F. Massicotte, “DIDroid: Android malware classification and characterization using deep image learning,” *ACM Int. Conf. Proceeding Ser.*, pp. 70–82, 2020, doi: 10.1145/3442520.3442522.

[8] A. Taha, A. H. Osman, and Y. S. Baguda, “An Intelligent Technique for Android Malware Identification Using Fuzzy Rank-Based Fusion,” *Technologies*, vol. 13, no. 2, Feb. 2025, doi: 10.3390/technologies13020045.

[9] T. Sanamontre, V. Visoottiviseth, and C. Ragkhitwetsagul, “Detecting Malicious Android Game Applications on Third-Party Stores Using Machine Learning,” *Lect. Notes Data Eng. Commun. Technol.*, vol. 202, pp. 238–251, 2024, doi: 10.1007/978-3-031-57916-5\_21.

[10] M. Ababneh, A. Al-Droos, and A. El-Hassan, “Modern Mobile Malware Detection Framework Using Machine Learning and Random Forest Algorithm,” *Comput. Syst. Sci. Eng.*, vol. 0, no. 0, pp. 1–10, 2024, doi: 10.32604/csse.2024.052875.