Intelligent Automation & Soft Computing
DOI: 10.32604/iasc.2023.030527
Article

Tech Science Press

Investigation of Android Malware Using Deep Learning Approach

V. Joseph Raymond[1,2,*] and R. Jeberson Retna Raj[1]

[1]Department of Computer Science and Engineering, Sathyabama Institute of Science and Technology, Chennai, 600119, Tamilnadu, India
[2]School of Computing, SRM Institute of Science and Technology, Chennai, 603203, Tamilnadu, India
*Corresponding Author: V. Joseph Raymond. Email: josephrv@srmist.edu.in
Received: 28 March 2022; Accepted: 05 May 2022

Abstract: In recent days the usage of android smartphones has increased extensively by end-users. There are several applications in different categories banking/finance, social engineering, education, sports and fitness, and many more applications. The android stack is more vulnerable compared to other mobile platforms like IOS, Windows, or Blackberry because of the open-source platform. In the Existing system, malware is written using vulnerable system calls to bypass signature detection important drawback is might not work with zero-day exploits and stealth malware. The attackers target the victim with various attacks like adware, backdoor, spyware, ransomware, and zero-day exploits and create threat hunts on the day-to-day basics. In the existing approach, there are various traditional machine learning classifiers for building a decision support system with limitations such as low detection rate and less feature selection. The important contents taken for building model from android applications like Intent Filter, Permission Signature, API Calls, and System commands are taken from the manifest file. The function parameters of various machine and deep learning classifiers like Nave Bayes, k-Nearest Neighbors (k-NN), Support Vector Machine (SVM), Ada Boost, and Multi-Layer Perceptron (MLP) are done for effective results. In our proposed work, we have used an unsupervised learning multilayer perceptron with multiple target labels and built a model with a better accuracy rate compared to logistic regression, and rank the best features for detection of applications and classify as malicious or benign can be used as threat model by online antivirus scanners.

Keywords: Android application; permissions; multilayer perception; relief scoring

1 Introduction

Android application-based vulnerabilities are exploited mostly through the static and dynamic approach. The recent finding suggests that a vulnerability will survive three updates, the second point third-party libraries are the major contributors to malicious payloads and the third point using encryption techniques for increasing the stealth techniques, and the fourth point detection tools can overshadow the payload [1]. In that paper, they have worked on the genesis of mobile application vulnerabilities, tools to address vulnerabilities, library screening strategies, and threats for validity. We have analyzed recent payloads taken from repositories and created a threat model for making a decision support system by ranking

malware using unsupervised learning. The demerit point with the machine learning approach is finding which best features to choose for the detection of malware. The balanced dataset always gives a better accuracy rate [2]. In my previous paperwork, we have detected malicious applications by executing payload in the emulator by emulating sensor events, acute binary translations, and hardware-assisted virtualizations using system call sequences [3,4]. In the existing approach, feature importance is ignored and the calculation of feature weights is irrelevant to the classification model. The state-of-the-art for weighting methods in features makes the decision support system more competitive [5]. The novel approach of recognizing malware by capturing the memory dump of the suspicious process can be represented as an image with Red-Green-Blue (RGB) combination using obfuscation and encryption techniques [6]. The research is carried out in three different phases gathering memory data, RGB Image representation, and visual feature extraction. In this paper, they have proposed a method for extracting features from metamorphic malware based on structural analysis obtained from theoretical and practical knowledge. They have also used dimensional reduction of features for finding a better accuracy rate [7]. The unique datasets are created by executing payloads in sandboxing environment taking into concern the important parameters taken from benchmarking datasets. In our proposed work, we have taken recent malicious android applications based on various families like adware, backdoor, Trojan, zero-day exploits, file indicators, etc. with a minimum of 1000 samples from each category taken from benchmark repositories and created a dataset. The dataset is implemented with supervised learning logistic regression and unsupervised learning multilayer perceptron with multiple regression and builds the model. This model utilizing machine learning calculations is proposed for recognizing the malware applications in cell phones dependent on the static malware examination procedure. The substance of an android application highlights is dissected, like authorizations, expectations, framework orders, and (Application Programming Interface) API calls that were removed from the application show document and source code [8]. The finding of event-leading normal noxious framework call codes in the framework call succession of a few malware families. At last, a malware recognition component is proposed dependent on the event of noxious framework call codes in the framework call grouping of an application [9]. We have ranked features using relief score and found a better accuracy rate for (Multi-Layer Perceptron) MLP over (Linear Regression) LR. Although the ranking of sensitive permission is always a major issue we have addressed taking a multiple-output layer which helps in analyzing documents or reports gathered during the initial assessment of android malware [10]. In the upcoming sections, we have related work in chapter 2, terminologies in chapter 3, and proposed methodology in chapter 4.

2 Related Works

Using binary classification approaches to examine Android malware is nearing a saturation point in machine learning research. In recent days, the use of clustering algorithms to analyse massive datasets, as well as feature selection to rate malware, may be the best option. When reviewing harmful android apps with Crowdoid, Sasidharan et al. [11] address how sensitive data is maintained. A Hidden Markov Model (HMM) has been employed by the researchers to observe state transitions and their related data An API sequence, suspicious API filtering, classification, and test and training data were obtained from the disassembly of APK files using IDA pro (Integrated Disassembly). In addition to Cuckoo sandboxing, Ubuntu is employed as the operating system for the execution of payload. Using elements like API, specific API calls, and categories based on signatures, Garcia et al. [12] were able to extract macro viral traits. To detect Android malware, Zhang et al. [13] employed opcode sequences to distribute learning datasets for training and testing on GPUs in the form of a convolutional neural network (CNN) and long short-term memory (LSTM) (LSTM). Receiver Operating Characteristic (ROC) curves were used to discover these rates. According to Yilin et al., portfolio optimization with return prediction employing multilayer perceptron (DMLP), LSTM and CNN, as well as SVR, RF and ARIMA [14] is explained.

According to Razgallah et al. [15], malware detections were studied, and they made recommendations for future research. There have also been other limits mentioned that may lead to a more accurate anti-malware system. There are sixteen recommendations based on both static and dynamic analysis. The feature extraction and classifier module of Repass Droid is used to monitor user-space activity. Google play, Drebin, Genome, AndroZoo, Contagio, VirusShare, Appchina, and PRAGuard datasets can be used to collect adware, backdoor Trojans, SMS attacks as well as signs of file enumeration. Android application and malware detection can be improved by using approaches such as reverse engineering, according to a survey by Sihag et al. [16]. Huang et al. [17] illustrate how Android applications may be made covert for the payload using reverse auditing and malicious injection. Code obfuscation and prevention are also on the list of easy-to-implement APK security features. Android hardening analysis, emulation, game theory and device binding have all been used to achieve this goal. Detecting malicious code in the system has also been a focus. Data-driven hybrid data assimilation is presented by Nellaivadivelu et al. [18] as a multilayer perceptron-based approach for capturing state variables in Decision Analysis (DA) situations. For the purposes of observation, context, and analysis, this research drew on the solver package and forecast model sources based on satellites optimised for software. It was used in the work of Millar et al. [19] to examine Android malware detectors. They concentrated on the most critical aspects for malware detection, these methods such as simple transformation, DSA transformation, and composite DSA transformation are compared to antivirus. Impossible features are extracted using bloom filters and a known malware family's signature database. As well as apktool and Java, they also used information from manifest files to test a range of obfuscators using various parameters, such as rebuild, indirections, renaming, and rearranging. Zhang et al. [20] used multi-view deep learning for the identification of Android malware that exploits zero-day vulnerabilities and categorises certain elements as risky to detect Drebin malware with 91% accuracy using a CNN (Convolutional Neural Network) technique. When Imtiaz et al. [21] investigated the features of Android malware, they found that Text-based CNN classification was successful and adaptable in detecting Android malware. k-Nearest Neighbors and protocol Droid approaches had an accuracy rate of 96.4 percent when applied to three independent data sets with larger samples in pre-static, static, and dynamic analysis. For Android malware detection and identification, Syed et al. developed a high-efficiency artificial neural network that exploited 78 percent of the system's vulnerabilities to achieve a 93 percent accuracy rate.

Table 1: Dataset information

Reference	Method	API calls	Intents	Permission	Limitation
[21]	K-mean clustering algorithm	Yes	Yes	Yes	Limited to static permissions used in the dataset
[22]	Multilayer perceptron	Yes	No	Yes	Used limited features in the dataset.
[23]	Artificial neural network	Yes	No	Yes	No intent and a low detection rate in the proposed model.
[24]	Deep belief network	Yes	No	No	Less accuracy rate in the proposed model,

3 Terminologies

3.1 Static Analysis

The probabilistic way to deal with identifying an Android malware from its decompiled source code. In this methodology, a prepared probabilistic classifier, for example, credulous Bayes or calculated relapse is utilized to anticipate whether the code level components, for example, API calls and consents are

pernicious or not. The proposed astatic examination-based Android malware identification instrument is called Drebin. Drebin extricates static elements from an application like equipment parts, consents, purposes, and API calls and utilizes Artificial Intelligence (AI) arrangement ways to deal with distinguishing if it is pernicious [23]. This instrument identifies Android malevolent applications from its API call reliance chart to find if the application is vindictive. The feature selection and extraction in this phase by decompiling the APK file, understanding Manifest and various resource assets.

3.2 Dynamic Analysis

In this methodology, an android application gathers all the framework call occasions from a gadget and sends them to a distant server through the cloud. Then, at that point, the server pre-processes this framework called information and utilizations k-implies grouping calculation to decide if the application is vindictive or not. The component for recognizing Android malware applications by dissecting the frequencies of framework calls that impact the conduct of an application [24]. In this methodology, a binary machine learning classifier is prepared with the frequencies of social framework calls delivered by known malware and good ware applications. The classifier can anticipate whether the frequencies of conduct framework calls by an obscure application relate to malware or not.

3.3 Dataset and Preliminaries

The android malware dataset collected CCCS-CIC-AndMal-2020, DREBIN, and MALGENOME which consists of the benign and malicious applications of different categories [25]. We have used AF Logical mobile forensics tool to extract features from the APK file and match them with the benchmark dataset for verifying the application. Tab. 1 shows the number of samples for each category along with the year of availability5 we also consider unique data sets collected from dynamic analysis through sandboxing approach.

The summary of existing malware detection studies helps in understanding the type of classifiers with limitations as shown in Tab. 2.

Table 2: Existing malware detection

Malware family	Number of samples taken for dataset	Total number of permissions in the dataset	Total number of sensitive feature permissions taken from Tab. 3
Adware	2081	51	17
Backdoor	1784	51	23
Dropper/Trojan	3044	51	22
File Infector	1670	51	10
Ransomware	1273	51	20
Scare ware	3066	51	21
SMS Attack	2300	51	19
Spyware	3054	51	26
Zero-Day	2300	51	27
Benign	3544	51	05

In our proposed approach we have overcome the weakness of the existing approach by using more permission vulnerable feature permissions and by fixing more target labels in MLP as shown in Tab. 3.

Table 3: Android application sensitive permissions

ID	Permissions	ID	Permissions
1	_WRITE_SMS	18	_PROCESS_OUTGOING_CALLS
2	_WRITE_SETTINGS	19	_MODIFY_PHONE_STATE
3	_WRITE_HISTORY_BM	20	_INTERNET
4	_WRITE_EXTERNAL_STORAGE	21	_INSTALL_PACKAGE
5	_WRITE_CONTACTS	22	_HARDWARE_TEST
6	_WRITE_APN_SETTINGS	23	_HARDWARE_TEST
7	_VIBRATE	24	_GET_ACCOUNTS
8	_USE_CREDENTIALS	25	_FACTORY_TEST
9	_SEND_SMS	26	_EXPAND_STATUS_BAR
10	_RESTART_ PACKAGE	27	_DIABLE_KEYGUARD
11	_RECEIVE_SMS	28	_DEVICE_POWER
12	_RECEIVE_BOOT_CMD	29	_CHANGE_WIFI_STATE
13	_READ_SMS	30	_CHANGE_NETWORK_STATE
14	_READ_PHONE_STATE	31	_CALL_PHONE
15	_READ_LOGS	32	_ACCESS_NETWORK_STATE
16	_READ_EXTERNAL_STORAGE	33	_ACCESS_LOCATION
17	_READ_CONTACTS	34	_ACCESS_GPS

The following system configurations are used for implementing decision support systems and building the model using MLP with multiple regression as shown in Tab. 4.

Table 4: Configuration details

Parameter	Value
Operating system	Windows 10 Professional 64 bit
CPU	Intel Core i7-9700, 8 Core, 12 MB Cache, 3.0 Ghz, 4.7 GHz
GPU	Turbo w/UHD Graphics 630
Hardware details	16 GB DDR4 2666 MHz UDIMM Non-ECC memory, 3.5 inch 1TB 7200 rpm SATA hard disk drive, M.2 256 GB PCIe NVMe Class 40 Solid state drive, Nvidia RTX 2080 Ti 11 GB graphic card
Platform/SDK	Google colab/Python/Orange visual tool

3.4 Malware Families

Zero-day exploits are the most expensive attacks where the attack is done inside software before the fix is released, is program protection in such kind of attack always challenging. Mostly this kind of attack is

protected using the Virtual Local Area Network (VLAN) and websites are protected with Security Socket Layer (SSL) in the android platform. The attack comes through advertisement-supported information that allows earning financial credits. Adware's are mostly available through third-party software that infects computers, servers, and mobile devices type of online attacks most of these safe and legitimate. The backdoor type of malware comes through the backdoor getting through unauthorized access or weak entry points are exploited and mitigated by the attacker and henceforth attack is done. In the last two years, many backdoor attacks are done by disgruntled employees inside the organization. As a countermeasure, the security check is done at the gateway, and the channel is secured. The Trojan as the name specifies, dropper drops payload in respective target place and launches event based on logical bomb used as a secret weapon to target victim. These are classified into various families and identified with the help of port numbers. This is an unusual type of attack where unexpected combinations of threats are exploited with kits to deliver file infectors into vulnerable systems. The users are driven to a malicious site that contains such kinds of kits. The stolen information is saved. (Dynamic Link Library) DLL file and uploaded to command and control servers. The ransomware makes victims have to pay some ransom for them to work again these types of malicious software when installed will stop some or a few events. This kind of payload will lock the computer screen and encrypt essential files. Scareware is also the type of ransomware claiming system caught by illegal activities. The SMS attack targets the victim's mobile phone with such kind creation and sends through unauthorized calls. They charge for SMS and generate revenue streams. We have to give more focus on these kinds of attacks ensuring protection from these kinds of attacks. Spyware is a type of attack mainly focused on surveillance attacks where data are monitored for commercial purposes which can breach security and confidential data can be mishandled. Data are comprised of spyware and gain active credentials [26].

4 Existing Classifiers

4.1 Logistic Regression

In the traditional approach, under classification, the primary classifier is linear or logistic regression based on the need of the user for training the model. The dataset is made by collecting datasets from repositories along with the own result obtained from the hybrid analysis. In the first part, logistic regression is supervised classification where our target variable is a discrete value stating whether the application is malicious or not called binary classification [27]. This model uses the sigmoid function given below in the equation.

$$g(z) = \frac{1}{1 + e^{\wedge} - z} \tag{1}$$

The logistic regression is categorized based on either low/high or high/low precision-recall where in the first case we reduce the false negative for sensitive data and in the latter reduce the false positives. The presence of android malware comes under binomial logistic regression where the presence is treated as '1' and not present as '0'.

$$\mathrm{y} = \{0, \text{if fail and } 1, \text{if pass}\} \tag{2}$$

The data has 'm' feature variables and 'n' observations and the matrix is represented as shown in Fig below.

$$\mathrm{X} = \begin{pmatrix} 1 & x11 & \cdots & x1m \\ \vdots & & \ddots & \vdots \\ 1 & xn1 & \cdots & xnp \end{pmatrix} \tag{3}$$

We can define conditional probabilities for two labels (0 and 1) for the $i^{th.}$

$$P(y_i = 1|x_i; \beta) = h(x_i) \quad (4)$$

$$P(y_i = 0|x_i; \beta) = 1 - h(x_i)$$

Here, y and h(x) present the vector and predicted response. Xj represents the observed values of the j^{th} feature.

$$\beta^j = \beta j - \alpha \sum_{i=1}^{n} (h(xi) - yi)xij \ldots \quad (5)$$

The learning is defined α and the value is set explicitly. The merits of using this approach are we need not take a learning rate, are always quick in execution, and get an appropriate numerical response. The demerits are a bit complex and look more like black-box testing. The important points to be noticed are independent variables and non-linear transformation. The dependent variables need not be normally distributed based on the binomial distribution. The homogeneity and errors need not be focused on in this type of approach based on large sample approximations. Tab. 3 shows the selection of permission for the detection of android malware. In this paper, we have created a dataset ensuring system calls, API calls, and permissions with recent exploits and zero-day vulnerabilities. The next step is the creation of a model using supervised learning.

4.2 Naïve Bayes Classifiers

This approach collection of algorithms where every pair classification is done independently, we divide it into the first feature matrix second response vector. The first part contains dependent features and later contains prediction in simple terms output. The equations given below explain the working principle of the classifier.

$$P(X|Y) \frac{P(Y|X)P(X)}{P(Y)} \quad (6)$$

where X and Y are events and checking whether P(Y)? 0 considering the probability of occurrence of X assuming Y is true termed as evidence. The prior and posterior probability have to be monitored. In our paper, we have used a Gaussian classifier. The demerit with this approach model will assume '0' as output in case the dataset has errors or missing values.

4.3 Support Vector Machine Classifiers

This approach is non-linear where we map high dimensional features with input data set. The outcome is non-probabilistic binary classification. The optimization of linear discriminant represents perpendicular distance. This classifier works on black-box testing where the input training data is compared with the output label and achieves the result.

4.4 K-Nearest Neighbours Classifiers

We have implemented a dataset using the K-Nearest Neighbours supervised machine algorithm one of the most essential classification algorithm mainly used for intrusion detection part of cyber security. This algorithm can be used for real-time data and henceforth is mostly suitable for hybrid analysis. Here we classify the data sets identified with the help of attributes.

4.5 Ada Boost Classifiers

An ensemble meta-algorithm that combines weak learners and adapts to the 'hardness' of each training sample. The AdaBoost (short for "Adaptive boosting") is a machine-learning algorithm, formulated by Yoav Freund and Robert Schapire. It can be used with other learning algorithms to boost their performance. It does so by tweaking the weak learners. AdaBoost works for both classification and regression.

5 Proposed Classifier

5.1 Multilayer Perceptron

The concept of perceptron comes with modern features in neural networks that explain binary classification '0' or '1' which can learn and solve complex problems [28]. This is purely taken on training data and power-based computation ability as shown in Fig. 1.

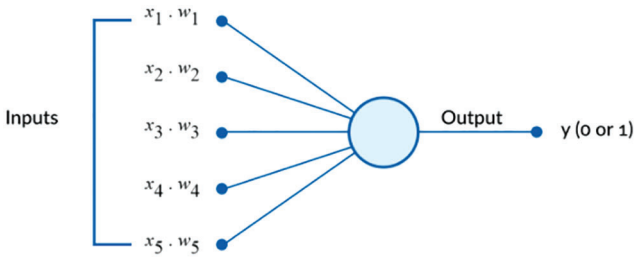


Figure 1: Black-box visualization of the proposed model

Multilayer perceptron comes along with additional perceptrons in layers for handling complex problems. This sends signals with different weights for different outputs. Fig. 2 given below shows three-layer MPL where the decision function will be a step function and the final output is binary.

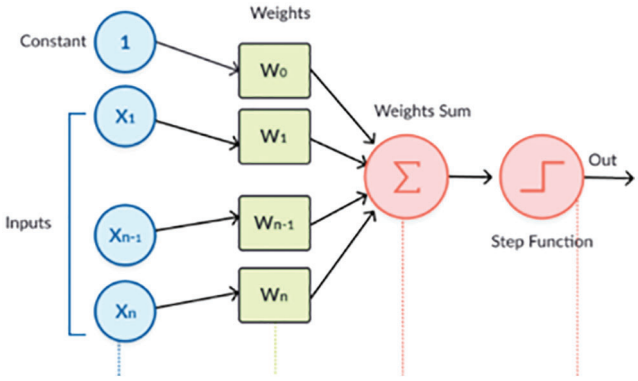


Figure 2: Multilayer Perceptron with different inputs and random weight

Fig. 3 below explains the perceptron learning process in step by step procedure. We take inputs to multiply with weights then compute the sum, add bias factor taking number 1 and multiply with weight feed sum through activation function and the final output will be perceptron. The backpropagation algorithm can be used by the term called "backward pass" which helps in tuning errors, calculating with square error and resulting in optimal weights is kind of hyper-parameter added through external entities which can create a huge impact on accuracy rate.

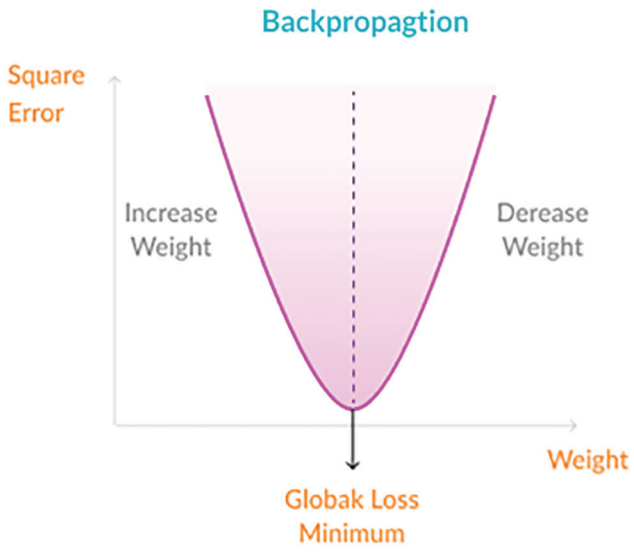


Figure 3: MLP with backpropagation

6 Proposed Methodologies

Authors are required to adhere to this Microsoft Word template in preparing their manuscripts for submission. It will speed up the review and typesetting process. In this paper, we have assigned random values to weights for increasing the accuracy rate and do random initialization with

$$\sqrt{(2 \div \text{size}[l-1])} \tag{7}$$

$$W^{[i]} = \text{np.random.randn}(\text{size\_l}, \text{size\_l} - 1) * \text{np.sqrt}(2 \div (\text{size\_1} - 1))$$

The approach above is an older technique followed for optimal results. We have improved by changing the equation as given below serves as a good initialization point and weights neither too big nor too less helping slow convergence shown in Eqs. (7) and (8). This reduces to minimum variance.

$$\sqrt{(2 \div \text{size}[l-1] + \text{size}[l])} \tag{8}$$

$$W^{[i]} = \text{np.random.randn}(\text{size\_l}, \text{size\_l} - 1) * \text{np.sqrt}(2 \div (\text{size\_1} - 1) + \text{size\_l})$$

The multiple output regression can have multiple output/target hidden layers which will be compared with features selected from the dataset. In our approach, we took a synthetic dataset taken from the repository Canadian university of cyber security and chose a data set to our requirement. The pseudo-code given below explains the implementation of a model for decision-making with a better accuracy rate.

MLP Multi-Output Algorithm

Step 1: Train the neural network.

Step 2: Implement Feedforward by feeding input layer, weight sum of input to the jth node in the hidden layer

Step 3: Calculate with the formula $\text{Net}_j = \sum W_{j:j} X_j + \theta_j$ and find aggregate input to the neuron. θ_j is the weighted value from the bias node always output value if 1.

Step 4: The bias node is considered as input to each hidden and output layer. This is for overcoming situations when input patterns are zero.

(Continued)

MLP Multi-Output Algorithm (continued)

Step 5: Add an appropriate activation function.

Step 6: The result from the activation function determines neuron output.

Step 7: We use the backpropagation algorithm when the activation function is differentiable and implement the sigmoid function $O_j = X_i = 1/(1 + e^{-Netj})$

Step 8: Error calculation and weight adjustment from Eqs. (7) and (8)

Step 9: Load dataset into Google Colab platform.

Step 10: Call make regression and pass number of samples, number of features, target and fix the random state.

Step 11: Call Sequential Function.

Step 12: Add Input and Output Layer.

Step 13: Assign a random weight to the activator function.

Step 14: Enumerate Fold and split train and test data from the dataset.

Step 15: Fit and evaluate the model.

Step 16: Print accuracy score, confusion matrix, and classification report

6.1 Evaluation Measures and Experimental Setup

The training and testing of the dataset are done by splitting data into 80% training and 20% testing data. We can have used label encoders for the target value we compute accuracy, precision, recall, and f-score for evaluating the proposed method, True Positive (TP) predicted the case to be in YES and present in it. False Positives (FP) predicted the case to be YES but NOT. True Negative (TN) predicted not to be in YES but NOT in it. In the last case, False Negative (FN) predicted the case not to be in YES but actually in YES. From the above results of MLP neural network implementation with backpropagation, we test the learning algorithms by comparing results with logistic regression in binary classification. We have 100 epochs for validating and testing the model. The first process is cross-validation by splitting data taken 5 folds based on features from the dataset. We select random samples and test on split test and train data. The results show the accuracy, precision, and recall for the model and achieve the optimal result of 96.5% for the neural network classifier compared to logistic regression as shown in Tab. 5 below achieved from Method MLP Multi-Output Algorithm. F-1 waited for the mean of precision and recall, the proportion of true positives among data classified as positive. Based on the comparison of existing

Table 5: Accuracy and classification report

Model	Accuracy	F-score	Recall	Precision
Multilayer perceptron[This work]	**96.5%**	**0.805**	**0.808**	**0.804**
k-NN	92.1%	0.744	0.749	0.746
Naïve Bayes	89.5%	0.554	0.549	0.587
SVM	90.5%	0.580	0.585	0.589
Ada Boost	96.3%	0.789	0.793	0.804
Logistic regression	83.3%	0.429	0.486	0.448

The input for confusion matrix results obtained from classification algorithms and output is selected data where we take into the number of instances, proportions of predicted and proportions of actual represents the target variable. Figs. 4–9 below shows the confusion matrix for various classifiers. It shows how many instances of normal applications and getting confused with the instance of malicious applications. The major strength of the proposed work is achieving 96.5% accuracy with larger data sets compared to the existing work considering zero-day exploits.

Predicted

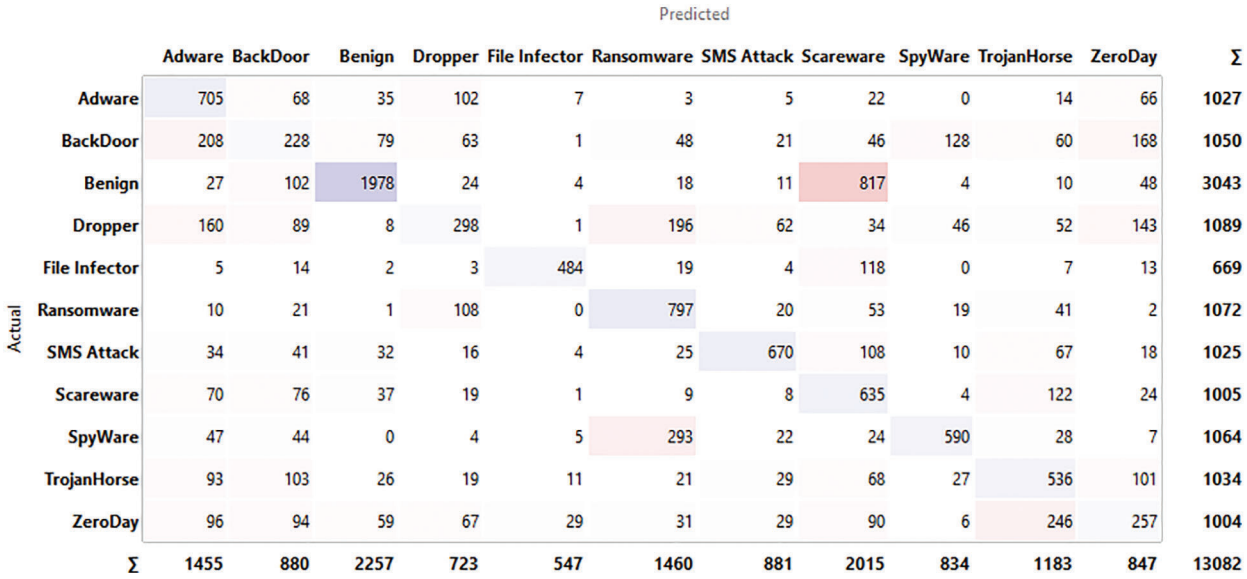
Actual	Adware	BackDoor	Benign	Dropper	File Infector	Ransomware	SMS Attack	Scareware	SpyWare	TrojanHorse	ZeroDay	Σ
Adware	875	27	29	16	13	4	3	8	2	10	40	1027
BackDoor	35	838	48	7	0	36	10	34	9	15	18	1050
Benign	29	48	2755	20	11	14	22	48	14	25	57	3043
Dropper	12	12	26	733	2	134	16	9	42	28	75	1089
File Infector	4	0	16	3	620	5	10	3	0	4	4	669
Ransomware	3	2	13	83	0	927	5	2	32	3	2	1072
SMS Attack	5	18	33	8	9	13	885	6	12	27	9	1025
Scareware	4	35	30	11	6	6	5	887	2	7	12	1005
SpyWare	1	4	11	11	0	84	12	5	904	13	19	1064
TrojanHorse	24	16	51	37	1	11	21	40	4	684	145	1034
ZeroDay	31	29	111	98	19	27	26	16	21	170	456	1004
Σ	1023	1029	3123	1027	681	1261	1015	1058	1042	986	837	13082

Figure 4: Confusion matrix for MLP

Predicted

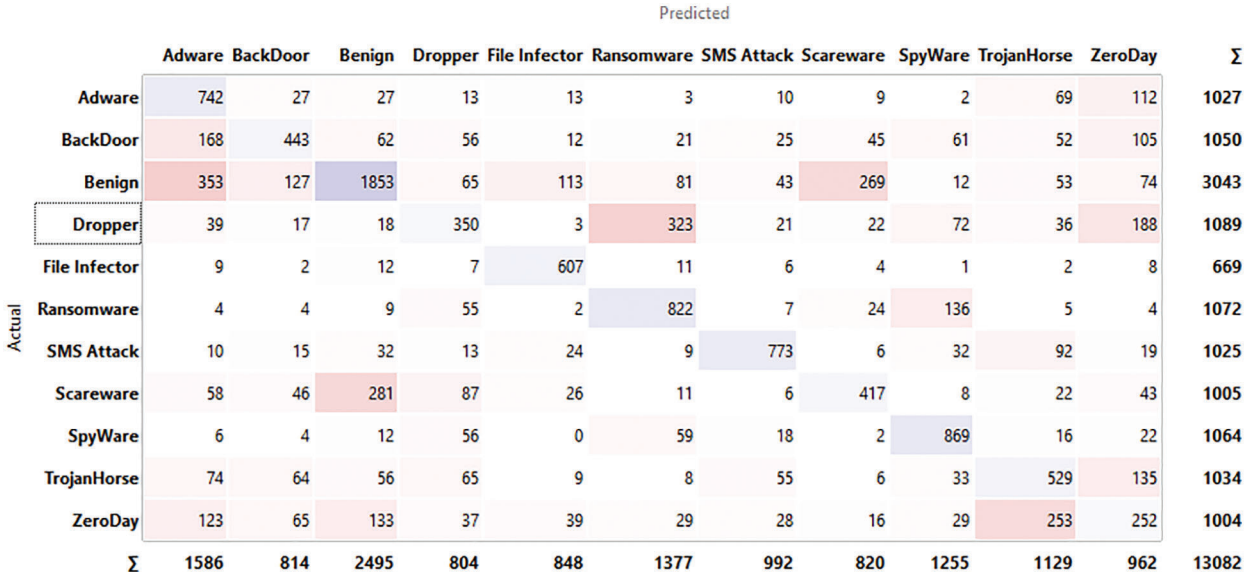
Actual	Adware	BackDoor	Benign	Dropper	File Infector	Ransomware	SMS Attack	Scareware	SpyWare	TrojanHorse	ZeroDay	Σ
Adware	658	27	110	3	1	29	14	10	136	20	19	1027
BackDoor	136	106	336	101	0	23	106	4	157	66	15	1050
Benign	16	16	2805	5	0	23	20	38	25	39	56	3043
Dropper	69	11	144	110	1	272	198	5	126	140	13	1089
File Infector	1	3	152	1	334	148	7	0	5	9	9	669
Ransomware	3	4	262	55	0	623	101	0	15	7	2	1072
SMS Attack	46	7	126	38	0	65	625	0	16	98	4	1025
Scareware	45	18	696	15	0	32	16	3	29	119	32	1005
SpyWare	18	8	103	10	2	263	101	1	454	102	2	1064
TrojanHorse	37	19	230	25	2	55	44	4	86	507	25	1034
ZeroDay	48	11	375	68	12	25	93	8	21	207	136	1004
Σ	1077	230	5339	431	352	1558	1325	73	1070	1314	313	13082

Figure 5: Confusion matrix for logistic regression



Actual \ Predicted	Adware	BackDoor	Benign	Dropper	File Infector	Ransomware	SMS Attack	Scareware	SpyWare	TrojanHorse	ZeroDay	Σ
Adware	705	68	35	102	7	3	5	22	0	14	66	1027
BackDoor	208	228	79	63	1	48	21	46	128	60	168	1050
Benign	27	102	1978	24	4	18	11	817	4	10	48	3043
Dropper	160	89	8	298	1	196	62	34	46	52	143	1089
File Infector	5	14	2	3	484	19	4	118	0	7	13	669
Ransomware	10	21	1	108	0	797	20	53	19	41	2	1072
SMS Attack	34	41	32	16	4	25	670	108	10	67	18	1025
Scareware	70	76	37	19	1	9	8	635	4	122	24	1005
SpyWare	47	44	0	4	5	293	22	24	590	28	7	1064
TrojanHorse	93	103	26	19	11	21	29	68	27	536	101	1034
ZeroDay	96	94	59	67	29	31	29	90	6	246	257	1004
Σ	1455	880	2257	723	547	1460	881	2015	834	1183	847	13082

Figure 6: Confusion matrix for Naïve Bayes

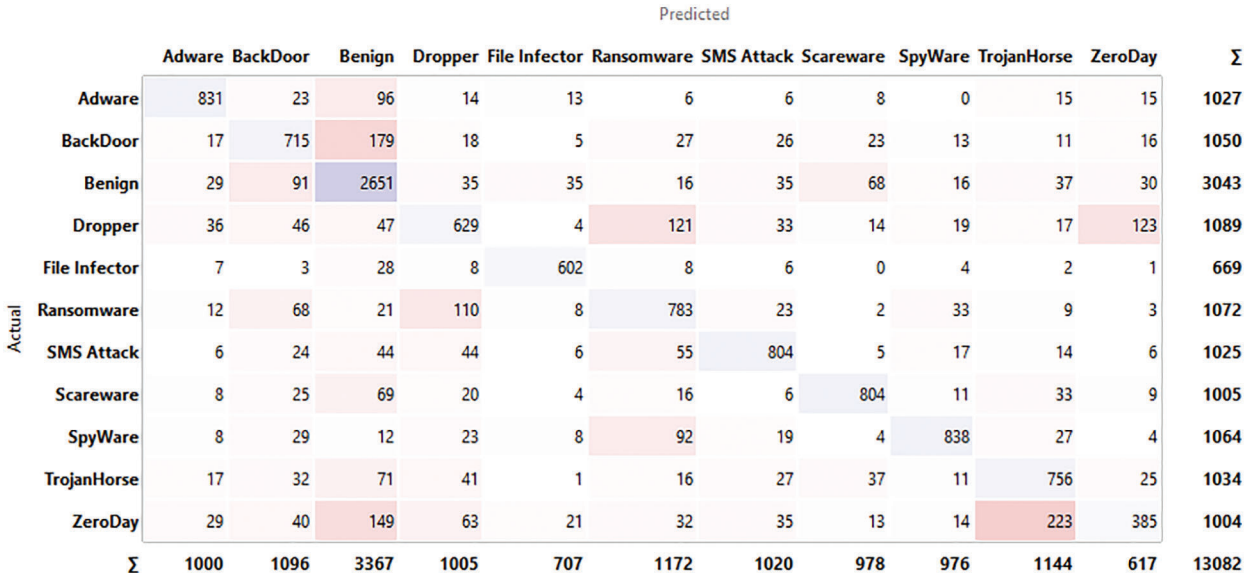


Actual \ Predicted	Adware	BackDoor	Benign	Dropper	File Infector	Ransomware	SMS Attack	Scareware	SpyWare	TrojanHorse	ZeroDay	Σ
Adware	742	27	27	13	13	3	10	9	2	69	112	1027
BackDoor	168	443	62	56	12	21	25	45	61	52	105	1050
Benign	353	127	1853	65	113	81	43	269	12	53	74	3043
Dropper	39	17	18	350	3	323	21	22	72	36	188	1089
File Infector	9	2	12	7	607	11	6	4	1	2	8	669
Ransomware	4	4	9	55	2	822	7	24	136	5	4	1072
SMS Attack	10	15	32	13	24	9	773	6	32	92	19	1025
Scareware	58	46	281	87	26	11	6	417	8	22	43	1005
SpyWare	6	4	12	56	0	59	18	2	869	16	22	1064
TrojanHorse	74	64	56	65	9	8	55	6	33	529	135	1034
ZeroDay	123	65	133	37	39	29	28	16	29	253	252	1004
Σ	1586	814	2495	804	848	1377	992	820	1255	1129	962	13082

Figure 7: Confusion matrix for SVM

The ROC curve below shows the tested model and its respective convex helps in determining the threshold and optimal classifier results shown in Fig. 10. The cost matrix is constructed based on the points at the border of concave regions. The multiple iteration's of testing and training using k-fold cross-validation. The default threshold will be considered as 0.5 and then target class probability.

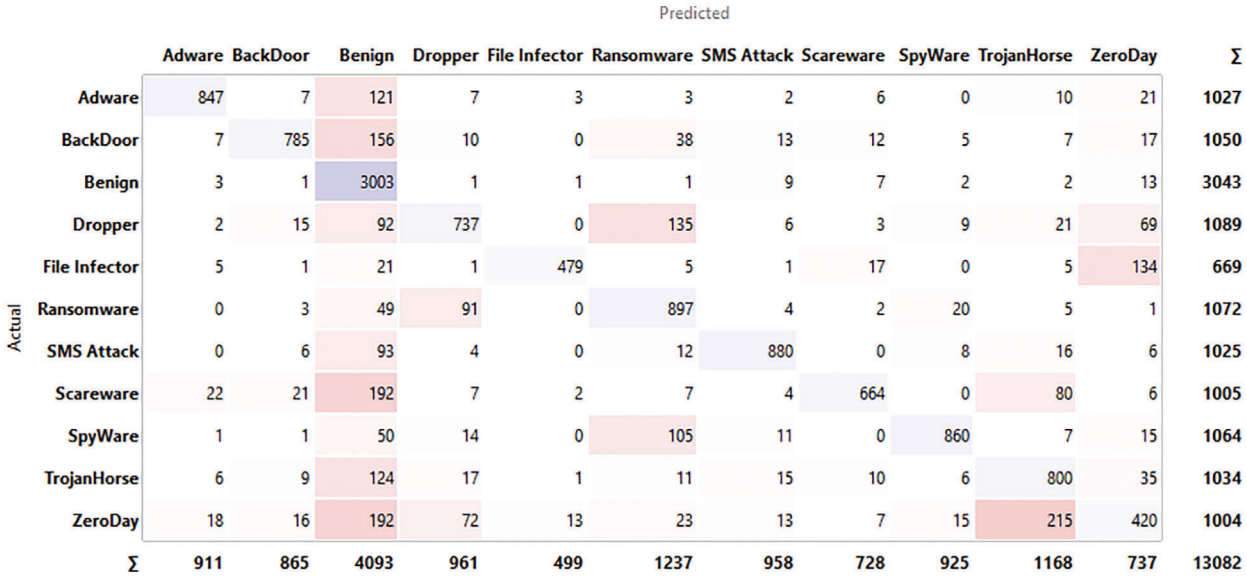
This is 2-dimensional visualization used for continuous and discrete attributes of various categories of malware families represented in the scatter plot shown in Fig. 11. The X-axis attributes contain a horizontal axis and the Y-axis attribute determines the position on the vertical axis like the color, size, and shapes of various attacks possible in android applications.



Predicted

Actual	Adware	BackDoor	Benign	Dropper	File Infector	Ransomware	SMS Attack	Scareware	SpyWare	TrojanHorse	ZeroDay	Σ
Adware	831	23	96	14	13	6	6	8	0	15	15	1027
BackDoor	17	715	179	18	5	27	26	23	13	11	16	1050
Benign	29	91	2651	35	35	16	35	68	16	37	30	3043
Dropper	36	46	47	629	4	121	33	14	19	17	123	1089
File Infector	7	3	28	8	602	8	6	0	4	2	1	669
Ransomware	12	68	21	110	8	783	23	2	33	9	3	1072
SMS Attack	6	24	44	44	6	55	804	5	17	14	6	1025
Scareware	8	25	69	20	4	16	6	804	11	33	9	1005
SpyWare	8	29	12	23	8	92	19	4	838	27	4	1064
TrojanHorse	17	32	71	41	1	16	27	37	11	756	25	1034
ZeroDay	29	40	149	63	21	32	35	13	14	223	385	1004
Σ	1000	1096	3367	1005	707	1172	1020	978	976	1144	617	13082

Figure 8: Confusion matrix for k-NN



Predicted

Actual	Adware	BackDoor	Benign	Dropper	File Infector	Ransomware	SMS Attack	Scareware	SpyWare	TrojanHorse	ZeroDay	Σ
Adware	847	7	121	7	3	3	2	6	0	10	21	1027
BackDoor	7	785	156	10	0	38	13	12	5	7	17	1050
Benign	3	1	3003	1	1	1	9	7	2	2	13	3043
Dropper	2	15	92	737	0	135	6	3	9	21	69	1089
File Infector	5	1	21	1	479	5	1	17	0	5	134	669
Ransomware	0	3	49	91	0	897	4	2	20	5	1	1072
SMS Attack	0	6	93	4	0	12	880	0	8	16	6	1025
Scareware	22	21	192	7	2	7	4	664	0	80	6	1005
SpyWare	1	1	50	14	0	105	11	0	860	7	15	1064
TrojanHorse	6	9	124	17	1	11	15	10	6	800	35	1034
ZeroDay	18	16	192	72	13	23	13	7	15	215	420	1004
Σ	911	865	4093	961	499	1237	958	728	925	1168	737	13082

Figure 9: Confusion matrix for AdaBoost

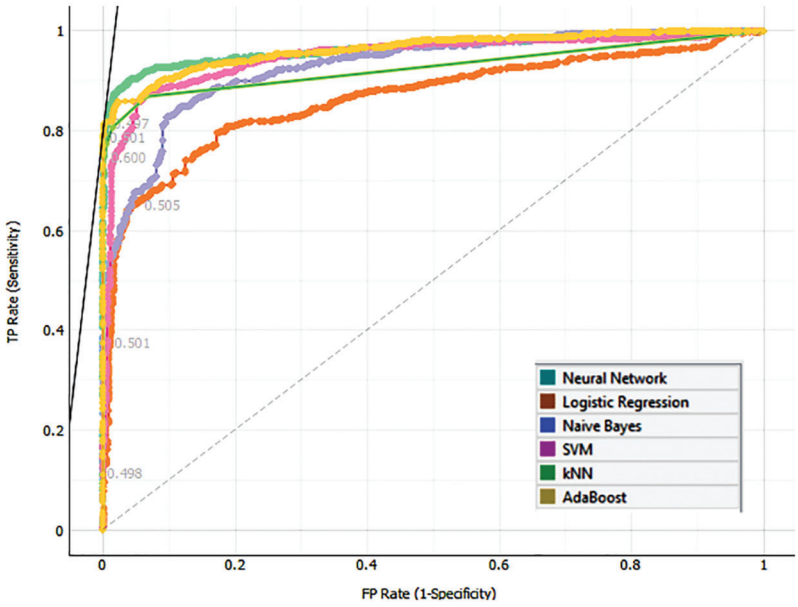


Figure 10: ROC curve

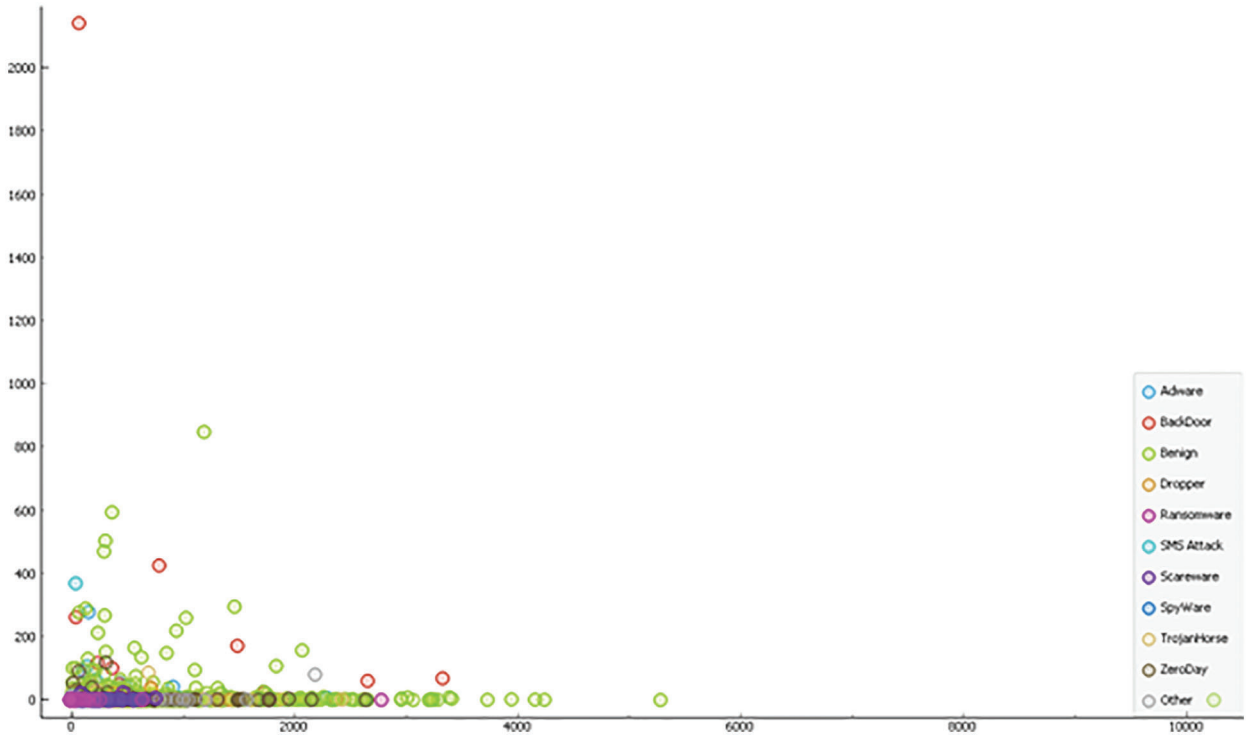


Figure 11: Scatter plot

6.2 Android Malware Feature Selection with Relief Scoring

The ranking of malware helps in reducing data and selecting attributes for making a better decision support system. The class-labeled datasets and scores attribute taking into consideration correlation with the class as shown in Fig. 12 below.

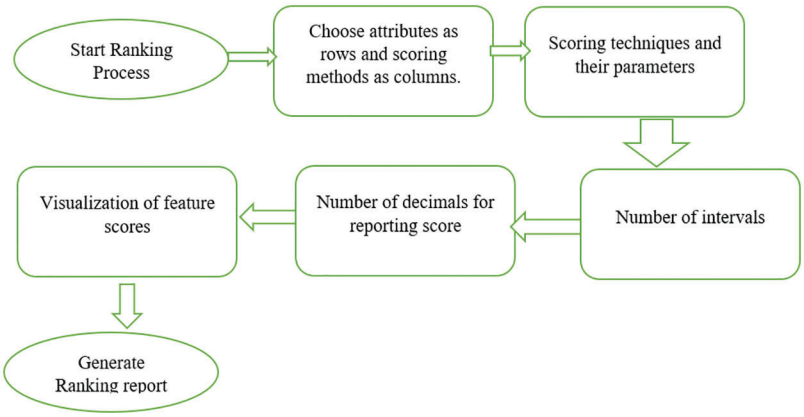


Figure 12: Ranking process

The filter-method technique used for feature selection for extracting the feature selection is mainly used for binary classification based on nearest neighbor instance pairs. The selection is done by nearest hit and nearest miss instance neighbors to scoring as shown in the formula given below.

$$W_i = W_i - (x_i - \text{nearHit}_i)^2 + (x_i - \text{nearMiss}_i)^2 \tag{9}$$

Fig. 13 below shows the top feature permission gathered from the dataset and its respective scores using this technique. The novelty of the proposed work is to ensure that the best sensitive permission features can be extracted from larger data sets with a better accuracy rate compared to the existing work.

	#	Info. gain	Gain ratio	ReliefF
N <WRITE_SMS>		0.470	0.235	0.012
N <WRITE_HISTORY_BM>		0.415	0.208	0.085
N <WRITE_APN_SETTINGS>		0.345	0.174	0.017
N <SEND_SMS>		0.327	0.164	0.038
N <WRITE_CONTACTS>		0.319	0.159	0.053
N <RECEIVE_BOOT_CMD>		0.296	0.150	0.037
C <Default : prevent app switches (S)>	2	0.291	0.328	0.126
N <RESTART_ PACKAGE>		0.280	0.141	0.024
C <GET_ACCOUNTS>	2	0.252	0.279	0.082
C <GET_APP_OPS_STATS>	2	0.244	0.252	0.155
C <ACCESS_GPS>	2	0.239	0.239	0.124
N <READ_SMS>		0.217	0.135	0.024
C <CHANGE_WIFI_STATE>	2	0.217	0.217	0.177
N <USE_CREDENTIALS>		0.216	0.113	0.018
C <DEVICE_POWER> (1)	2	0.203	0.243	0.147
C <email.ACCESS_PROVIDER>	2	0.183	0.187	0.103
C <GET_TASKS>	2	0.139	0.145	0.069
C <HARDWARE_TEST>	2	0.139	0.249	0.068
N <RECEIVE_SMS>		0.135	0.191	0.002
C <CHANGE_NETWORK_STATE>	2	0.135	0.254	0.052
C <DIABLE_KEYGUARD>	2	0.130	0.146	0.101
C <DIAGNOSTIC>	2	0.128	0.189	0.160
C <CAMERA>	2	0.126	0.138	0.107
C <INTERNET>	2	0.124	0.124	0.113
C <FACTORY_TEST>	2	0.122	0.241	0.047
C <INSTALL_PACKAGE>	2	0.098	0.122	0.135
C <READ_PHONE_STATE>	2	0.098	0.148	0.117
C <GET_PACKAGE_SIZE>	2	0.093	0.103	0.080
C <Default : power device on or off (S)>	2	0.087	0.121	0.071

Figure 13: Best permission features from the model based on ranking

7 Conclusion and Future Work

In this approach, we have selected the best features that can be used for ranking android malware using multilayer perceptron with multiple regression and achieved better accuracy compared with logistic regression. The dataset collected from repositories is taken into dataset according to different malware categories that are seen to be more vulnerable in recent days. The unobserved android malware is identified and provides the solution for threats like zero-day exploits which cannot be tracked by anti-virus scanners using a signature-based approach. The technical part covers analyzing the android malware using sandboxing environment preparing reports from dynamic analysis and giving that as input for the deep learning model. The ranking method will enhance the system's ability to identify the threat and can be used in online antivirus scanners. We have tested with recent payloads collected from benchmarking dataset. As the functions of each application are increasingly powerful it has become mandatory for us to protect the user from vulnerable threats as we know that most of the Applications on the Android platform are not encrypted. As the next generation moving towards smart cities where most users will be using the android application for various purposes like banking, finance, fitness and health, social applications the possibility of threats might increase in the case of unencrypted applications as well as weaker applications. This might be one of the major challenges while establishing IoT platforms where most devices are connected to the internet resulting in the breaking of integrity and confidentiality. Our proposed work leading to the threat model can suggest or help in decision-making for users while installing an application from not trusted resources. In future work, we take binary samples to convert into an image with RGB combinations and classify them using the CNN approach. In the industry, as many employees might be using android phones there might be a possibility a disgruntled person can float vulnerable applications and cause social engineering attack out automation approach will help organizations to track sensitive permissions and report to the forensics team for further studies. In addition that can implement Gradient Boosting (XG-Boost) model compared with traditional classifiers and justify with better accuracy for larger samples under unsupervised learning.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] J. Gao, L. Li, P. Kong, T. F. Bissyandé and J. Klein, "Understanding the evolution of android app vulnerabilities," *IEEE Transactions on Reliability*, vol. 70, no. 1, pp. 238–250, 2021.

[2] R. Surendran., T. Thomas and S. Emmanuel, "On existence of common malicious system call codes in android malware families," *IEEE Transactions on Reliability*, vol. 70, no. 1, pp. 218–230, 2021.

[3] V. J. Raymond and R. J. R. Raj, "Reversing and auditing of android malicious applications using sandboxing environment," *International Journal of Electronic Security and Digital Forensics*, vol. l, no. 12, pp. 386–396, 2020.

[4] K. Shibija and R. V. Joseph, "A machine learning approach to the detection and analysis of android malicious apps," in *Proc. IEEE Int. Conf. on Computer Communication and Informatics (ICCCI)*, Tamilnadu, India, pp. 1–4, 2018.

[5] L. Cai, Y. Li and Z. Xiong, "JOWMDroid: Android malware detection based on feature weighting with joint optimization of weight-mapping and classifier parameters," *Computers & Security*, vol. 100, no. 7, pp. 102086–102098, 2021.

[6] A. S. Bozkir., E. Tahillioglu, M. Aydos and I. Kara, "Catch them alive: A malware detection approach through memory forensics, manifold learning and computer vision," *Computers & Security*, vol. 103, no. 1, pp. 102166–102185, 2021.

[7] Y. T. Ling, N. Sani, M. T. Abdullah and N. A. Hamid, "Structural features with nonnegative matrix factorization for metamorphic malware detection," *Computers and Security*, vol. 103, no. 2, pp. 102216, 2021.

[8] R. Surendran, T. Thomas and S. Emmanuel, "On existence of common malicious system call codes in Android malware families," *IEEE Transactions on Reliability*, vol. 70, no. 1, pp. 248–260, 2020.

[9] P. R. Varshini, S. Baskar, M. Varatharajan and S. Sadhana, "Tuning rules for fractional order pid controller using data analytics," *Intelligent Automation and Soft Computing*, vol. 33, no. 3, pp. 1787–1799, 2022.

[10] M. Cai, Y. Jiang, C. Gao, H. Li and W. Yuan, "Learning features from enhanced function call graphs for Android malware detection," *Neurocomputing*, vol. 423, no. 7, pp. 301–307, 2021.

[11] S. K. Sasidharan and C. Thomas, "ProDroid-An Android malware detection framework based on profile hidden Markov model," *Pervasive and Mobile Computing*, vol. 23, no. 4, pp. 101336–101389, 2021.

[12] D. E. García and N. D. C. García, "Optimal feature configuration for dynamic malware detection," *Computers & Security*, vol. 103, pp. 102250, 2021.

[13] N. Zhang, Y. A. Tan, C. Yang and Y. Li, "Deep learning feature exploration for Android malware detection," *Applied Soft Computing*, vol. 102, no. 1, pp. 107069–107077, 2021.

[14] Y. Ma, R. Han and W. Wang, "Portfolio optimization with return prediction using deep learning and machine learning," *Expert Systems with Applications*, vol. 165, no. 3, pp. 113973–113992, 2020.

[15] A. Razgallah, R. Khoury, S. Hallé and K. Khanmohammadi, "A survey of malware detection in Android apps: Recommendations and perspectives for future research," *Computer Science Review*, vol. 39, no. 3, pp. 100358, 2021.

[16] V. Sihag, M. Vardhan and P. Singh, "A survey of android application and malware hardening," *Computer Science Review*, vol. 39, no. 1, pp. 100365, 2021.

[17] L. Huang, H. Leng, X. Li, K. Ren and J. Song, "A data-driven method for hybrid data assimilation with multilayer perceptron," *Big Data Research*, vol. 23, no. 10, pp. 100179–100188, 2021.

[18] G. Nellaivadivelu, F. Troia and M. Stamp, "Black box analysis of android malware detectors," *Array*, vol. 6, no. 2, pp. 100022–100034, 2020.

[19] S. Millar, N. McLaughlin, J. D. Rincon and P. Miller, "Multi-view deep learning for zero-day Android malware detection," *Journal of Information Security and Applications*, vol. 58, no. 3, pp. 102718–102736, 2021.

[20] N. Zhang, Y. A. Tan, C. Yang and Y. Li, "Deep learning feature exploration for Android malware detection," *Applied Soft Computing*, vol. l, no. 102, pp. 107069–107076, 2020.

[21] S. Imtiaz, S. Rehman, A. R. Javed, Z. Jalil and W. S. Alnumay, "DeepAMD: Detection and identification of Android malware using high-efficient deep artificial neural network," *Future Generation Computer Systems*, vol. 115, no. 5, pp. 844–856, 2020.

[22] D. S. Keyes, B. Li, G. Kaur, A. H. Lashkari and F. Gagnon, "EntropLyzer: Android malware classification and characterization using entropy analysis of dynamic characteristics," in *Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS)*, Canada, pp. 1–18, 2021.

[23] D. J. Wu, C. Mao, T. Wei, H. Lee and K. P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," in *Proc. IEEE Seventh Asia Joint Conf. on Information Security*, Tokyo, Japan, pp. 62–69, 2012.

[24] P. Chan and W. Song, "Static detection of Android malware by using permissions and API calls," in *Proc. IEEE Int. Conf. on Machine Learning and Cybernetics*, Koyoma, Japan, pp. 82–87, 2014.

[25] Z. Yuan, Y. Lu and Y. Xue, "Droiddetector: Android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114–123, 2018.

[26] Z. Yuan, Y. Lu, Z. Wang and Y. Xue, "Droid-sec: Deep learning in android malware detection," in *Proc. of the ACM Special Interest Group on Data Communication*, United States, pp. 371–372, 2014.

[27] J. Kinyua and L. Awuah, "AI/ML in security orchestration, automation and response: Future research directions," *Intelligent Automation and Soft Computing*, vol. 28, no. 2, pp. 527–545, 2021.

[28] K. Xu, Y. Li, R. H. Deng and K. Chen, "Deeprefiner: Multi-layer android malware detection system applying deep neural networks," in *Proc. 2018 IEEE European Symp. on Security and Privacy (EuroS&P)*, London, United Kingdom, pp. 473–487, 2018.