ISSN 2278-3091
Volume 9, No.5, September - October 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse307952020.pdf
https://doi.org/10.30534/ijatcse/2020/307952020

Automated Feature Selection using Boruta Algorithm to Detect Mobile Malware

Che Akmal Che Yahaya, Ahmad Firdaus, Salwana Mohamad, Ferda Ernawan, Mohd Faizal Ab Razak
Faculty of Computing, College of Computing and Applied Sciences, University Malaysia Pahang, 26600 Pekan, Pahang Darul Makmur
pcn18003@stdmail.ump.edu.my, firdausza@ump.edu.my, salwanamohamad@ump.edu.my, ferda@ump.edu.my, faizalrazak@ump.edu.my

ABSTRACT
The usage of android system is rapidly growing in mobile devices. Android system might also incur severe different malware dangers and security threats such as infections, root exploit, Trojan, and worms. The malware has potential to compromise and steal the private data, classified data, instant messages, private business contacts, and confidential schedule. Malware detection is needed due to the malware continuously evolve rapidly. This research proposed automated feature selection using Boruta algorithm to detect the malware. The proposed method adopts machine learning prediction and optimizes the selecting features in order to reduce the model of machine learning complexity. Boruta algorithm is used to select features automatically for assisting the machine learning. The experimental results show that the proposed method is able to reach 99.73% accuracy in machine learning classification.

Key words: Android; static analysis; machine learning; features.

1. INTRODUCTION

Nowadays, people utilize smartphone to communicate and assists them in daily activities, from normal until important tasks. There is various type of operating system (OS) built for smartphone; for instance, Android, IOS, Window, Blackberry and Symbian. Among all these OSes, [1] stated that android is the largest installed platform and growing fast which placed first among others. Consequently, there is a rapid increase in the amount of malware targeting Android smartphones since it is the most popular OS [2].

Malware is an acronym for malicious software - that executes malicious activities such as, secretly accessing private information, causes damage to the OS, or control the system. Therefore, in order to detect malware, security practitioners adopt two types of analyses; dynamic and static. Dynamic analysis is an analysis that execute and monitor the malware behavior [3]. However, dynamic analysis is unable to discover some parts of the code that execute outside of the monitoring range. Apart from that, dynamic analysis is a high resource-consuming analysis, which requires a high specification of hardware [3]. Therefore, static analysis is another alternative for the researcher to detect malware. It is an analysis that studies the malware without executing it. Additionally, this analysis is able to discover the malware that would behave under unusual conditions which is much more accurate [4]. This is due to static analysis examine overall parts of a program including parts that excluded in dynamic analysis, and able to detect unknown malware with machine learning [5].

However, it is important to consider the challenges in machine learning. One of it is the features concern. Higher number of features will decrease the machine learning Android malware detection system performance. This is because by having a large number of features will increase the dimension of search space for the problem. Consequently, this will cause the problem to suffer from Curse of dimensionality [6]. Therefore, there is a need to find a suitable algorithm to select the best among many number of features. Hence, the main contributions in this technical paper are as below:

a) applied Boruta algorithm to select the best features automatically. In author's knowledge, to date, only this paper adopted this algorithm in selecting features in detecting malware for Android platform.
b) this paper discovers the effectiveness of Boruta algorithm in detecting malware.
c) utilized the public dataset [7] which contains hundreds of features that consists of different categories. These features are from 1260 malware and 2539 normal @ benign applications. By using this dataset, Boruta have a wider choice of features to select from and decrease it. From the features selected from the Boruta, it will increase the machine learning detection rate.
d) evaluates the results to measure the Boruta effectiveness to detect malware.

2. RELATED WORK

This section introduces the types of malware analysis, and then followed by the machine learning information, types of feature selection, Boruta algorithm and comparison with previous studies.

2.1 Malware analysis (dynamic and static)

There are two types of analysis to detect malware; dynamic and static. ***Table 1*** shows the comparison between these two analyses.

Table 1: Differences between static and dynamic analysis in malware detection

Type of analysis	Strength	Weakness
Dynamic	1) May detect anomaly in malware	1) Extensive duration 2) High resource utilization
Static	1) Instant detection (fast) 2) May detect malware in anomaly in malware with machine learning prediction.	1) Need to update the input data if needed.

Dynamic approaches are convenient, yet it does have disadvantages such as high efficiency and profitability. In addition, it is difficult to detect malware types that able to disguise their suspicious activities during analysis. Conversely, static analysis is an approach which evaluates the malicious program without running the applications.

Static (offline) training means that we basically have a big data store and we train our model exactly once before it's used for a long time. Offline training requires less monitoring or monitoring of dataset training work than online training. The downside of the static is that it still requires monitoring at the input data at the inference time, if needed. If our distribution of inputs changes and our model has not adapted, we may end up with a bleak prediction. Hence, it is important to have big data enough to support detection for a long time. However, we able to update the input data anytime as needed as well. Furthermore, with machine learning assist, static analysis able to predict anomaly in malware, similar with dynamic analysis.

2.2 Machine learning

Machine learning (ML) is a category of algorithms that allows software applications to become more accurate in predicting outcomes without being explicitly programmed. The basic premise of machine learning is to build algorithms that able to train itself according to the input data given and then use statistical analysis to predict an output. If the new data available, it will update he output. The machine learning classification used in this experiment are Random forest, J48 and GLM. However, machine learning needs relevant features to predict the output efficiently [8][9].

2.3 Types of feature selection methods

The selection of features, also known as a selection of variables, selection of attributes, or selection of variable subsets, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction. Feature selection techniques are used for several reasons:

a) Simplification of models to make them easier for researchers/users to interpret.
b) Shorter training time
c) Avoiding the curse of dimensionality
d) Enhanced generalization by reducing over-fitting (formally, reduction of variance)

Accordingly, this paper implements Boruta algorithm to select the best features.

2.4 Boruta Algorithm

Boruta is a feature selection algorithm that works as a wrapper built around a random forest classification algorithm. It captures all the important or interesting features of the dataset concerning the output variable. The technique performs a top-down search for important features by comparing the significance of the original attributes with the significant randomly attainable attributes, assessing the use of their permuted duplicates (shadows), and gradually eliminating unimportant features to balance the test. Attributes or features that are significantly better than shadows are recognized as confirmed. On each iteration, shadows are re-created and the algorithm stops when only confirmed features are left. In short, Boruta algorithm use strategy a top-down search for significant features by contrasting original attributes important and important reachable at random, evaluated utilizing their permuted duplicates, and dynamically wiping out irrelevant features to stabilize that test. Therefore, this research uses the Boruta algorithm as a feature selection [10].

2.5 Comparison of the existing methods

This section compares the features selection and classification in previous research.

Table 2 lists that, to date, only this experiment used Boruta algorithm in selecting the relevant features in detecting mobile Android, which none researchers apply it previously. In our experiment, Boruta algorithm selects the best features automatically and investigate its effectiveness in selecting the best features.

Table 2: Features selection and classification in previous research

References	Year	Features Selection algorithm	Classification algorithm
[11]	2017	Simulated annealing	Beta Classifier

			and Metaclassifier
[12]	2018	Genetic algorithm	Naïve bayes, Random forest, Multilayer perceptron, J48 and Functional tree
[13]	2019	Permission ranking-based features selection approach	Random Forest
[14]	2017	Correlation-based feature selection (CFS), Chi square (CHI), Information gain (IG), ReliefF (RF) and one wrapper method with a Linear SVM classifier (WR)	SVM Classifier
This paper	Curren t year	Boruta algorithm	J48

3. METHODOLOGY

This section provides the methodology in conducting our experiment. Figure 1 consists of three phases; 1) literature review; 2) feature selection and machine learning classification; and 3) result.

As **Error! Reference source not found.** depicts the first phase involves all the review processes that includes identifying the problem statement, objective, scope, existing research, technique to select features and software involves in the experiment. These processes is to achieve the idea and to decide which algorithms need to use to select the best features.

The second phase is where we used R to import and read the dataset. This paper used a public dataset from a reliable paper entitled *"Droidfusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection"* [7]. This dataset consists of 1260 malware and 2539 benign @ normal samples. The 215 features in this dataset is based from 3799 Android applications (.apk). The features are form multiple categories, namely; 1) permission; 2) API calls; 3) directory path; and 4) string.

As machine learning need to reduce the complexity of its model, this phase used R to execute the Boruta algorithm to decrease the number of features automatically. This phase runs Boruta until it reaches a state that there are no features deemed important. *Table 3* tabulates 215 features before Boruta algorithm.

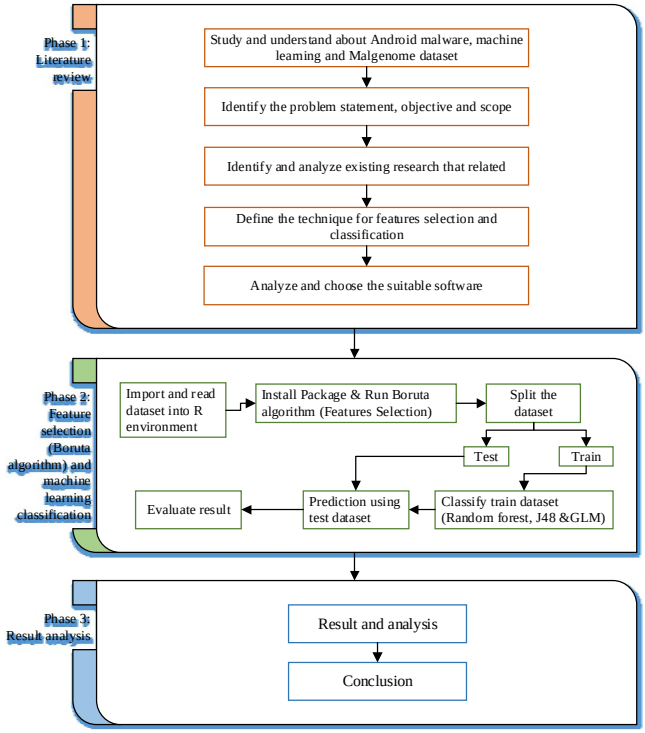

Figure 1: Methodology of Boruta algorithm

Table 3: List of 215 features before Boruta algorithm

transact	Ljava.lang.Class.forName	READ_SYNC_STATS	DexClassLoader	intent.action.RUN	STATUS_BAR
bindService	TelephonyManager.getSimSerialNumber	WRITE_HISTORY_BOOKMARKS	WRITE_CALENDAR	SecretKey	Ljavax.crypto.Cipher
onServiceConnected	CAMERA	DISABLE_KEYGUARD	PROCESS_OUTGOING_CALLS	CLEAR_APP_CACHE	MODIFY_PHONE_STATE
ServiceConnection	CALL_PHONE	READ_LOGS	BIND_DEVICE_ADMIN	ACCESS_FINE_LOCATION	android.intent.action.PACKAGE_RESTARTED
android.os.Binder	android.intent.action.SEND	RECORD_AUDIO	CHANGE_WIFI_MULTICAST_STATE	SET_WALLPAPER_HINTS	READ_INPUT_STATE
READ_SMS	onBind	getCallingPid	MASTER_CLEAR	Context.bindService	READ_EXTERNAL_STORAGE
attachInterface	android.content.pm.Signature	MODIFY_AUDIO_SETTINGS	android.intent.action.PACKAGE_DATA_CLEARED	MessengerService	Ljava.lang.Object.getClass
WRITE_SMS	READ_SYNC_SETTINGS	android.intent.action.PACKAGE_REPLACED	FLASHLIGHT	ACCESS_NETWORK_STATE	SET_ORIENTATION
TelephonyManager.getSubscriberId	AUTHENTICATE_ACCOUNTS	android.intent.action.TIMEZONE_CHANGED	android.intent.action.BATTERY_LOW	android.content.pm.PackageInfo	DEVICE_POWER
Ljava.lang.Class.getCanonicalName	INTERNET	BROADCAST_STICKY	SET_ALARM	BIND_ACCESSIBILITY_SERVICE	EXPAND_STATUS_BAR
Ljava.lang.Class.getMethods	PackageInstaller	Runtime.exec	RECEIVE_MMS	INTERNAL_SYSTEM_WINDOW	GET_TASKS
android.intent.action.BOOT_COMPLETED	ACCESS_LOCATION_EXTRA_COMMANDS	android.intent.action.PACKAGE_ADDED	divideMessage	SET_TIME_ZONE	GLOBAL_SEARCH

Ljava.lang.Class.getField	HttpUriRequest	MOUNT_UNMOUNT_FILESYSTEMS	WRITE_CALL_LOG	Process.start	GET_PACKAGE_SIZE
READ_PHONE_STATE	remount	android.intent.action.ACTION_POWER_DISCONNECTED	WRITE_PROFILE	MOUNT_FORMAT_FILESYSTEMS	SET_PREFERRED_APPLICATIONS
Landroid.content.Context.unregisterReceiver	android.telephony.SmsManager	Ljava.lang.Class.getDeclaredClasses	WRITE_USER_DICTIONARY	CLEAR_APP_USER_DATA	android.intent.action.PACKAGE_CHANGED
GET_ACCOUNTS	RECEIVE_BOOT_COMPLETED	android.intent.action.PACKAGE_REMOVED	BIND_INPUT_METHOD	UPDATE_DEVICE_STATS	
SEND_SMS	android.intent.action.ACTION_POWER_CONNECTED	BLUETOOTH_ADMIN	READ_SOCIAL_STREAM	IRemoteService	
Landroid.content.Context.registerReceiver	findClass	android.os.IBinder	REORDER_TASKS	android.intent.action.SET_WALLPAPER	
getBinder	WRITE_CONTACTS	IBinder	defineClass	BROADCAST_WAP_PUSH	
Ljava.lang.Class.cast	.system.app	WRITE_SECURE_SETTINGS	PERSISTENT_ACTIVITY	android.intent.action.CALL_BUTTON	
chmod	Ljava.lang.Class.getResource	WRITE_SETTINGS	ProcessBuilder	INJECT_EVENTS	
createSubprocess	WRITE_SYNC_SETTINGS	Ljavax.crypto.spec.SecretKeySpec	android.intent.action.SCREEN_ON	ACCESS_SURFACE_FLINGER	
Ljava.net.URLDecoder	android.intent.action.TIME_SET	android.intent.action.BATTERY_OKAY	READ_USER_DICTIONARY	SET_PROCESS_LIMIT	
WRITE_APN_SETTINGS	android.intent.action.SEND_MULTIPLE	READ_CONTACTS	WRITE_SOCIAL_STREAM	ADD_VOICEMAIL	
TelephonyManager.getDeviceId	ACCESS_WIFI_STATE	Binder	SET_TIME	INSTALL_LOCATION_PROVIDER	
RECEIVE_SMS	URLClassLoader	SUBSCRIBED_FEEDS_READ	mount	SET_ACTIVITY_WATCHER	
Ljava.lang.Class.getDeclaredField	BLUETOOTH	READ_CALL_LOG	System.loadLibrary	TelephonyManager.getCallState	
HttpGet.init	WAKE_LOCK	SUBSCRIBED_FEEDS_WRITE	CHANGE_COMPONENT_ENABLED_STATE	VIBRATE	
Ljava.lang.Class.getPackage	SYSTEM_ALERT_WINDOW	BATTERY_STATS	ACCESS_MOCK_LOCATION	Runtime.getRuntime	
abortBroadcast	TelephonyManager.getSimCountryIso	RECEIVE_WAP_PUSH	DUMP	CHANGE_CONFIGURATION	
ClassLoader	chown	PathClassLoader	CALL_PRIVILEGED	BROADCAST_SMS	
TelephonyManager.getLine1Number	NFC	KILL_BACKGROUND_PROCESSES	DELETE_PACKAGES	BIND_WALLPAPER	
getCallingUid	READ_HISTORY_BOOKMARKS	ACCESS_COARSE_LOCATION	READ_FRAME_BUFFER	BROADCAST_PACKAGE_REMOVED	
USE_CREDENTIALS	HttpPost.init	android.intent.action.ACTION_SHUTDOWN	WRITE_GSERVICES	TelephonyManager.isNetworkRoaming	
MANAGE_ACCOUNTS	TelephonyManager.getNetworkOperator	Runtime.load	ACCOUNT_MANAGER	TelephonyManager.getSimOperator	
android.telephony.gsm.SmsManager	Ljava.lang.Class.getClasses	android.intent.action.SENDTO	KeySpec	WRITE_EXTERNAL_STORAGE	
.system.bin	BIND_REMOTEVIEWS	SET_WALLPAPER	sendDataMessage	android.intent.action.CAMERA_BUTTON	
Ljava.lang.Class.getMethod	READ_PROFILE	android.intent.action.NEW_OUTGOING_CALL	android.intent.action.CALL	android.intent.action.REBOOT	
RESTART_PACKAGES	READ_CALENDAR	CHANGE_NETWORK_STATE	BIND_APPWIDGET	sendMultipartTextMessage	
INSTALL_PACKAGES	CHANGE_WIFI_STATE	REBOOT	android.intent.action.SCREEN_OFF	BIND_VPN_SERVICE	

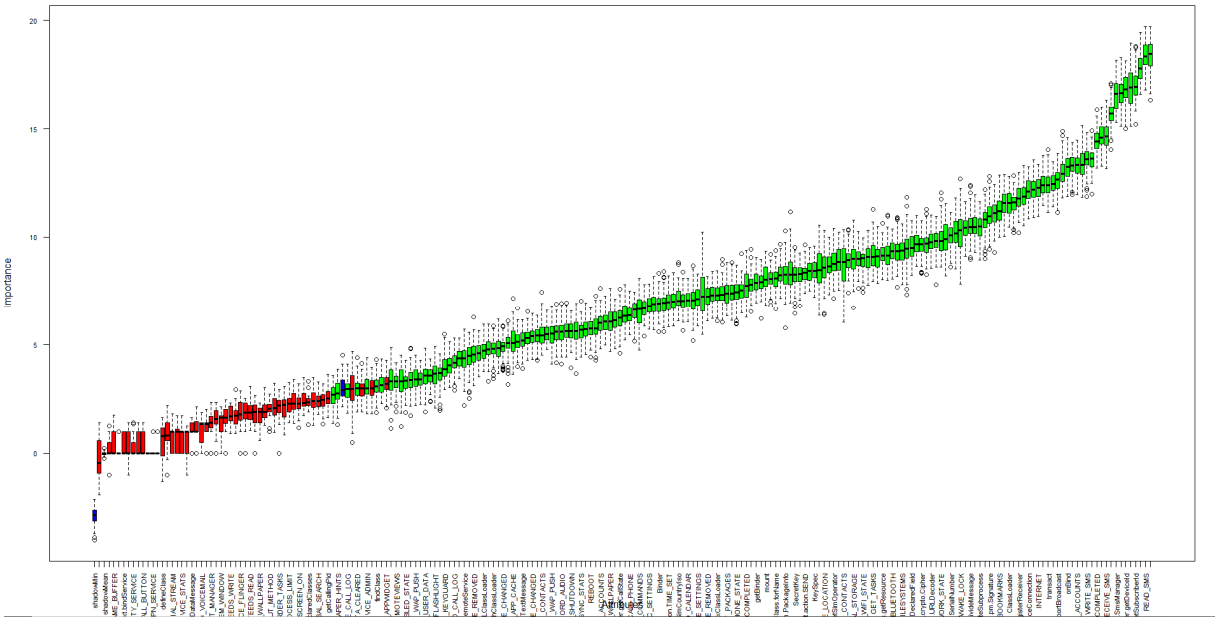

Figure 2: 1st round of Boruta

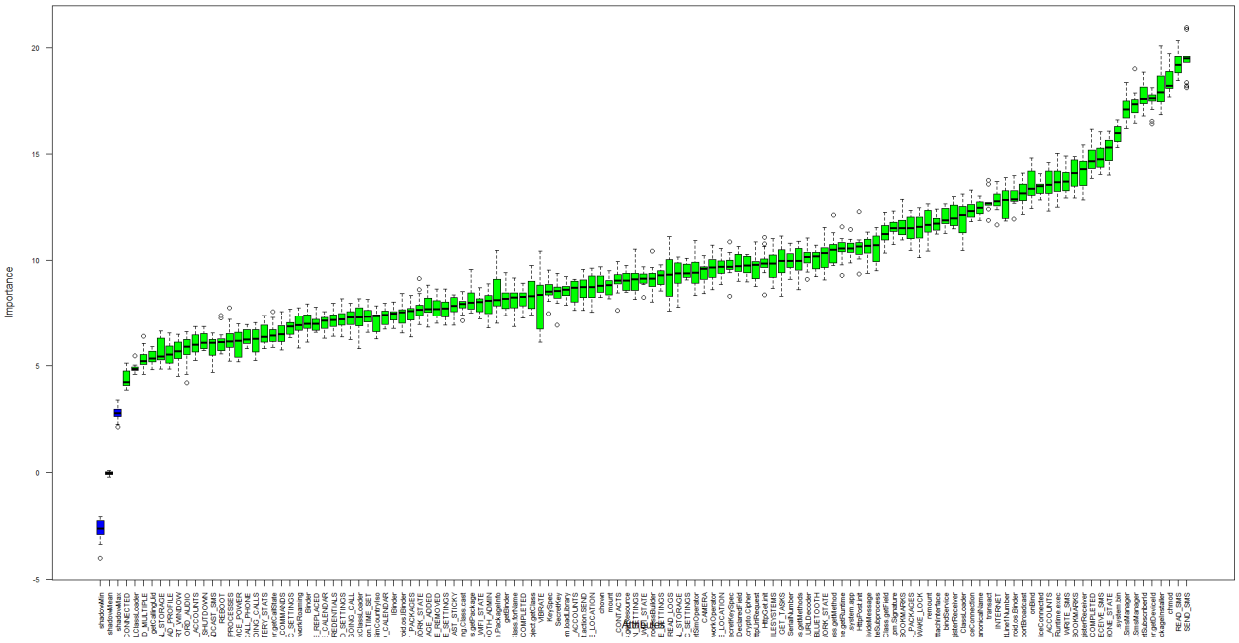

Figure 3: 8th round of Boruta

Table 4: List of 124 features after Boruta algorithm

transact	RECEIVE_SMS	PackageInstaller	WRITE_HISTORY_BOOKMARKS	DexClassLoader
bindService	Ljava.lang.Class.getDeclaredField	ACCESS_LOCATION_EXTRA_COMMANDS	READ_LOGS	WRITE_CALENDAR
onServiceConnected	HttpGet.init	HttpUriRequest	RECORD_AUDIO	PROCESS_OUTGOING_CALLS
ServiceConnection	Ljava.lang.Class.getPackage	remount	MODIFY_AUDIO_SETTINGS	divideMessage
android.os.Binder	abortBroadcast	android.telephony.SmsManager	android.intent.action.PACKAGE_REPLACED	ProcessBuilder
READ_SMS	ClassLoader	RECEIVE_BOOT_COMPLETED	BROADCAST_STICKY	mount
attachInterface	TelephonyManager.getLine1N	android.intent.action.ACTION	Runtime.exec	System.loadLibrary

	umber	_POWER_CONNECTED		
WRITE_SMS	getCallingUid	.system.app	android.intent.action.PACKAGE_ADDED	KeySpec
TelephonyManager.getSubscriberId	USE_CREDENTIALS	Ljava.lang.Class.getResource	MOUNT_UNMOUNT_FILESYSTEMS	SecretKey
Ljava.lang.Class.getCanonicalName	MANAGE_ACCOUNTS	WRITE_SYNC_SETTINGS	android.intent.action.PACKAGE_REMOVED	ACCESS_FINE_LOCATION
Ljava.lang.Class.getMethods	android.telephony.gsm.SmsManager	android.intent.action.TIME_SET	BLUETOOTH_ADMIN	ACCESS_NETWORK_STATE
android.intent.action.BOOT_COMPLETED	.system.bin	android.intent.action.SEND_MULTIPLE	android.os.IBinder	android.content.pm.PackageInfo
Ljava.lang.Class.getField	Ljava.lang.Class.getMethod	ACCESS_WIFI_STATE	IBinder	TelephonyManager.getCallState
READ_PHONE_STATE	RESTART_PACKAGES	URLClassLoader	WRITE_SECURE_SETTINGS	VIBRATE
Landroid.content.Context.unregisterReceiver	INSTALL_PACKAGES	BLUETOOTH	WRITE_SETTINGS	Runtime.getRuntime
GET_ACCOUNTS	Ljava.lang.Class.forName	WAKE_LOCK	Ljavax.crypto.spec.SecretKeySpec	BROADCAST_SMS
SEND_SMS	TelephonyManager.getSimSerialNumber	SYSTEM_ALERT_WINDOW	READ_CONTACTS	TelephonyManager.isNetworkRoaming
Landroid.content.Context.registerReceiver	CAMERA	TelephonyManager.getSimCountryIso	Binder	TelephonyManager.getSimOperator
getBinder	CALL_PHONE	chown	BATTERY_STATS	WRITE_EXTERNAL_STORAGE
Ljava.lang.Class.cast	android.intent.action.SEND	READ_HISTORY_BOOKMARKS	KILL_BACKGROUND_PROCESSES	Ljavax.crypto.Cipher
chmod	onBind	HttpPost.init	ACCESS_COARSE_LOCATION	READ_EXTERNAL_STORAGE
createSubprocess	android.content.pm.Signature	TelephonyManager.getNetworkOperator	android.intent.action.ACTION_SHUTDOWN	Ljava.lang.Object.getClass
Ljava.net.URLDecoder	READ_SYNC_SETTINGS	READ_PROFILE	android.intent.action.NEW_OUTGOING_CALL	DEVICE_POWER
WRITE_APN_SETTINGS	AUTHENTICATE_ACCOUNTS	READ_CALENDAR	CHANGE_NETWORK_STATE	GET_TASKS
TelephonyManager.getDeviceId	INTERNET	CHANGE_WIFI_STATE	REBOOT	

This phase runs from 1st to 8th round of Boruta features. ***Figure 2*** shows the result of the Boruta in 1st round, while the following figure (***Figure 3***) depicts the 8th round. The number of times to reach this state is 8th times. Once the process is finished, the Boruta successfully reduced the features from 215 to 124. ***Table 4*** lists those 124 features. With these 124 features, this phase trains three machine learning classifiers (Random forest, J48 and GLM) to construct a model. Then, in order to test the model either it able to detect unknown malware, we test this model with the data that excluded from the training part. In phase three, we executed the classification process. The machine learning classifier used are Random Forest, J48, and GLM.

4. EXPERIMENTAL RESULTS

This section discussed the results of machine learning classification, derived from the Boruta algorithm. The following figures depict the results of the classification.

```
Confusion Matrix and Statistics

          Reference
Prediction   B   S
         B 741   3
         S   0 369

               Accuracy : 0.9973
                 95% CI : (0.9921, 0.9994)
    No Information Rate : 0.6658
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9939

 Mcnemar's Test P-Value : 0.2482

            Sensitivity : 1.0000
            Specificity : 0.9919
         Pos Pred Value : 0.9960
         Neg Pred Value : 1.0000
             Prevalence : 0.6658
         Detection Rate : 0.6658
   Detection Prevalence : 0.6685
      Balanced Accuracy : 0.9960

       'Positive' Class : B
```

Figure 4: Random Forest algorithm result

Random Forest algorithm is one of the methods used in this research for the classification process. Figure 4 shows the result of the Random Forest classifier. algorithm predicts the class of the data. There are two classes which are 'B' for Benign and 'S' it shows the confusion matrix and statistics for the Random Forest algorithm using the test dataset. There are 1140 data inside the test dataset. Random Forest's accuracy is 0.9973.

```
Confusion Matrix and Statistics

          Reference
Prediction   B   S
         B 730  13
         S  11 359

               Accuracy : 0.9784
                 95% CI : (0.9681, 0.9861)
    No Information Rate : 0.6658
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.9515

 Mcnemar's Test P-Value : 0.8383

            Sensitivity : 0.9852
            Specificity : 0.9651
         Pos Pred Value : 0.9825
         Neg Pred Value : 0.9703
             Prevalence : 0.6658
         Detection Rate : 0.6559
   Detection Prevalence : 0.6676
      Balanced Accuracy : 0.9751

       'Positive' Class : B
```

Figure 5: J48 algorithm result

Figure 5 shows the confusion matrix and statistics for the J48 algorithm using the Test dataset. There are 1140 data inside the test dataset. J48 accuracy is 0.9784.

```
> # Confusion matrix for threshold of 0.5
> table(train$class, predictTrain > 0.5)

    FALSE TRUE
  B  1761    0
  S     0  898
>
> # Sensitivity
> 898/(898+0)
[1] 1
>
> #Specificity
> 1761/(1761+0)
[1] 1
>
>
> # Confusion matrix for threshold of 0.2
> table(train$class, predictTrain > 0.2)

    FALSE TRUE
  B  1761    0
  S     0  898
>
> # Sensitivity
> 898/(898+0)
[1] 1
>
> #Specificity
> 1761/(1761+0)
[1] 1
> #prediction on test set
> predictTest = predict(classLog, type = "response", newdata = test)
Warning message:
In predict.lm(object, newdata, se.fit, scale = 1, type = if (type ==  :
  prediction from a rank-deficient fit may be misleading
> table(test$class,predictTest >= 0.3)

    FALSE TRUE
  B   758   20
  S     3  359
>
> #Accuracy
> (359+758)/(758+20+3+359)
[1] 0.9798246
```

Figure 6*:* GLM algorithm result

Error! Reference source not found. shows the GLM confusion matrix algorithm using the test dataset. There are 1140 data in the test dataset. The precision is 0.9798. The GLM sensitivity and specificity are calculated manually based on the confusion matrix that is generated automatically by the system. Table below list overall accuracies with different machine learning classifiers.

Table 5: Result accuracy each algorithm

Classifiers	Accuracy
Random forest	0.9973
J48	0.9784
GLM	0.9798

Based on ***Table 5*** both classifier which is Random Forest and J48 decision tree, it shows that Random Forest has the higher accuracy which is 0.99 while J48 Decision Tree is 0.97. In conclusion, Random Forest is the best compare to J48 and GLM.

5. CONCLUSION

Malware continuously evolve rapidly and it also compromise and steal the private data especially in Android system. Malware detection with high accuracy is needed to detect its malicious activities. This paper presented automated feature selection using boruta algorithm and machine learning method to detect and predict mobile malware. The proposed the optimal features selection is able to reduce the model of machine learning complexity. The experimental results show that the proposed Boruta algorithm provides automatically feature selection for increasing the machine learning detection. The result shows that random forest successfully achieved 99.73% accuracy in machine learning detection. The proposed scheme able to select the best and relevant features efficiently.

ACKNOWLEDGEMENT

This work is supported by Ministry of Higher Education (MOHE) for Fundamental Research Grant Scheme (FRGS-RACER) with grant number RDU192607, RACER/1/2019/ICT02/UMP//5, and Universiti Malaysia Pahang, under the Grant IBM Centre of Excellence (COE)(IBM2000), RDU180337.

REFERENCES

1. F. M. Gotz, S. Stieger, and U. D. Reips. **Users of the main smartphone operating systems (iOS, Android) differ only little in personality**, *PLoS ONE*, vol. 12, no. 5, pp. 1–18, 2017.
2. M. F. A. Razak, N. B. Anuar, R. Salleh, and A. Firdaus. **The rise of ‘“malware”’: Bibliometric analysis of malware study**, *Journal of Network and Computer Applications*, vol. 75, pp. 58–76, 2016.
3. W. Enck. **Defending Users against Smartphone Apps: Techniques and Future Directions**, *Proceedings of the 7th international conference on Information Systems Security*, pp. 49–70, 2011.
4. C. A. Castillo. **Android Malware Past, Present, and Future**, McAfee White Paper, Mobile Security Working Group, 2011. [Online]. Available: http://www.mcafee.com/us/ resources/white-papers/wp-android-malware-past-present-future.pdf.
5. K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro. **The Evolution of Android Malware and Android Analysis Techniques**, *ACM Computing Surveys (CSUR)*, vol. 49, no. 4, pp. 1–41, 2017.
6. A. Kumar, K. S. Kuppusamy, and G. Aghila. **FAMOUS: Forensic Analysis of Mobile Devices Using Scoring of**

application permissions, *Future Generation Computer Systems*, vol. 83, pp. 158–172, 2018.

7. 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, 2019.
8. A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab. **A review on feature selection in mobile malware detection**, *Digital Investigation*, vol. 13, pp. 22–37, 2015.
9. A. Firdaus, N. B. Anuar, M. F. A. Razak, I. A. T. Hashem, S. Bachok, and A. K. Sangaiah. **Root Exploit Detection and Features Optimization: Mobile Device and Blockchain Based Medical Data Management**, *Journal of Medical Systems*, vol. 42, no. 112, pp. 1-23, 2018.
10. M. B. Kursa, A. Jankowski, and W. R. Rudnicki. **Boruta-A system for feature selection**, *Fundamenta Informaticae*, vol. 101, no. 4, pp. 271–285, 2010.
11. M. Hassen and P. K. Chan. **Scalable function call graph-based malware classification**, *Proceedings of the 7th ACM Conference on Data and Application Security and Privacy*, pp. 239–248, 2017.
12. A. Firdaus, N. B. Anuar, A. Karim, and M. F. A. Razak. **Discovering optimal features using static analysis and a genetic search based method for Android malware detection**, *Frontiers of Information Technology and Electronic Engineering*, vol. 19, no. 6, pp. 712–736, Jun. 2018.
13. R. Kumar, X. Zhang, R. U. Khan, and A. Sharif. **Research on data mining of permission-induced risk for android IoT devices**, *Applied Sciences*, vol. 9, no. 2, pp. 1–22, 2019.
14. M. Z. Mas'ud, S. Sahib, M. F. Abdollah, S. R. Selamat, and C. Y. Huoy. **A comparative study on feature selection method for N-gram mobile malware detection**, *International Journal of Network Security*, vol. 19, no. 5, pp. 727–733, 2017.