Android Malware Classification and Optimisation Based on BM25 Score of Android API

1st Rahul Yumlembam
Computer and Information Sciences
Northumbria University
Newcastle upon Tyne, UK
rahul.yumlembam@northumbria.ac.uk

2nd Biju Issac
Computer and Information Sciences
Northumbria University
Newcastle upon Tyne, UK
bissac@ieee.org

3rd Longzhi Yang
Computer and Information Sciences
Northumbria University
Newcastle upon Tyne, UK
longzhi.yang@northumbria.ac.uk

4th Seibu Mary Jacob
School of Computing, Engineering & Digital Technologies
Teesside University
Middlesbrough, UK
s.jacob@tees.ac.uk

***Abstract*—With the growth of Android devices, there is a rise in malware applications affecting these networked devices. Android malware classification is an important task in ensuring the security and privacy of Android devices. One promising approach to this problem is to capture the difference in the usage of API in benign and malware applications through the BM25 (Best Matching 25) scoring function by calculating the BM25 score of each API (Application Program Interface). A linear regression model is fitted using the BM25 score to select the 1000 most important APIs using the feature importance weight of the linear regression model. The selected API's BM25 score and the Permission and Intents of an application are used to train Naive Bayes, Random Forest, Decision Tree, Support Vector Machine, and CNN (Convolutional Neural Network) for classification. To illustrate the effectiveness of using the BM25 score of APIs for malware classification, we train the optimised Particle Swarm Optimisation (PSO) based Machine learning and Deep Learning algorithms using Permission and Intents features with and without the BM25 score. Experiments show that the BM25 score improves the result. Overall, this study demonstrates the potential of using the BM25 score of API calls, in combination with Permissions and Intents, as a valuable tool for Android malware classification.**

***Index Terms*—Android Malware, Optimisation, Machine Learning, Convolutional Neural Network, Android API**

I. Introduction

The number of Android mobile users is growing rapidly, and it is the most used Operating System in the mobile market as of 2021. There is the rapid emergence of new malware in the market and according to AV-test in 2019 alone, 3.18 million new Android malware emerged. Such exposure makes the development of new malware detection techniques vital. Android malware can potentially steal sensitive and personal information. Last year, there were more than 300,000 reported Android malware infections via Droppers on Google Play Store [1]. In recent years, machine learning and deep learning techniques have been used for Android malware detection. One challenge in using APIs for malware detection is the high dimensionality of the feature space. To address this issue, we proposed using term weighting schemes, the popular BM25 (Best Match 25) score, to weight the importance of each API (Application Program Interface) in the feature vector. The main contributions of our research are as follows:

- We propose an approach for classifying Android malware using BM25 score of APIs in an application, along with Permission and Intent.
- The performance of classification algorithms is improved by finding the optimal hyper-parameter using PSO.
- We analyse the performance of the proposed methods on two benchmark datasets, i.e., CICMalDroid 2020 and Drebin and compare our approach with the state-of-the-art systems and found out that BM25 score approach has improved the accuracy on both the benchmark datasets.

II. Related works

Deep learning has hailed an enormous success in the domain of image and text processing. Applying deep learning to identify android malware is currently becoming one of the active research areas. There are three main ways of android malware analysis, namely, static analysis, dynamic analysis and hybrid analysis. In static analysis, requested Permissions, Intents, API calls, and App components are the common features used for analysis. The work reported in [3] uses permissions, sensitive API calls, Intent, and App components as features. Similar features are also utilized in [4] in combination with hardware features, strings, code patterns, certificate info, and payload information. Both approaches use deep belief networks to extract abstract feature representation and SVM (Support Vector Machine) to classify malware and benign application. Residual LSTM (Long Short Term Memory) is used along with features similar to [4] in [6], [7]. Single static features are used in [8], [9]. Sequential android API calls were input as features to train a CNN (Convolutional Neural Network) and use LSTM to classify in [8]. In [9], Permissions used in an android application is used as features passed to an embedding layer and trained an LSTM for classification. Pektas and Acarmen [12] extracted execution paths in terms

of op-codes and fed into an embedding layer to capture the relationship between two correlated op-code, a CNN is used as a backbone with LSTM to output the probability of an application being Benign or Malware. In dynamic analysis, features in consideration include network flow information, app actions, the sequence of API calls, CPU and memory information. In [13] and [5], in addition to static features, dynamic information like action_sendnet, which sends data over the network, are also employed as features. Both methods use Deep belief Networks to extract abstract features. In [14] sequence of API call and app actions are generated in a stateful manner. The multi-modal deep learning method is used to classify between benign and malware applications.

III. Overview of Classification

To classify an Android application into malware and benign, the following steps are performed as explained below: (1) Decompilation (2) Feature Extraction and (3) Malware classification.

Decompilation: The Android applications which are written in Java are compiled into .class by Java compiler. The .class file is then converted into the .dex file using Android SDK. The generated .dex along with all the resources such as images, video files, image files, XML files etc. are packaged into APK (Android Application Package) using the Android assets packaging tool. To extract the APIs, Permission, and Intent, we decompile each APK using APKtool [10] and extract the Smali files and manifest files. Smali files are human-readable format of the .dex file which is also known as Dalvik executables. A sample Smali file is shown in figure 1. Android manifest file describes vital information about an application, such as code namespace, permission to access the functionality of the device or other application, and intent of the application, which is a messaging object used to request an action from app components.

Feature Extraction: The APIs used in an application from the Smali files are extracted along with the frequency of each API used in the application. An example is shown in figure 1 from the Smali code segment. $SmsManager \rightarrow sendTextMessage$ along with $application/Checker \rightarrow scheduleChecking$ will be extracted as an API (other API's in the code segment is not shown due to space constraint). The extracted APIs are then assigned a unique global identifier. The unique global identifier is similar across all the applications. In this paper for BM25 score-based classification, the corresponding BM25 score for each APIs with respect to each of the application is calculated using the frequency of each API used in the application. Using BM25 score of each API as features and linear regression as the model, the top 1000 APIs are selected based on the importance weight of each API.

Malware classification: The generated BM25 score of the selected APIs is concatenated along with the Permission and Intent used in an application. The concatenated vector is used as a feature to train Machine learning algorithms to classify an application into benign and malware.

```
.method startSendingMessages ()V
.....
.....
.....
invoke-virtual/range {vo.. v5},
Landroid/telephony/SmsManager;->sendTextMessage(Ljava/lang/String..
.line 652
iget-object v1, p0, Lcom/software/application/Actor;->mContext:
Landroid/content/Context;
invoke-static {v1}, Lcom/software/application/Checker;->scheduleChecking
(Landroid/content/Context;)V
.line 653 return-void
.end method
```

Fig. 1. Sample Smali code of an Application

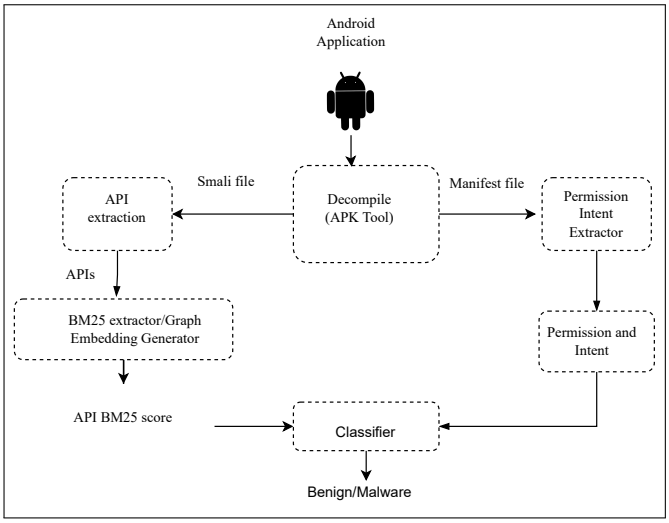

Fig. 2. Overall Architecture

IV. Proposed Method

In this section, we briefly discuss about the dataset used, the features used and the classification algorithms.

A. Dataset

The datasets used for this work are CICMalDroid 2020 [16] and Drebin dataset [17]. CICMalDroid 2020 contains 17,341 samples and as reported in the original dataset website, only 13,077 samples ran successfully without any errors. In our experiment, we were able to extract features from 15,848 application (3696 Benign and 12,152 Malware). This dataset also includes other static and dynamic features along with the apks. Drebin dataset contains 5,560 malware APKs, the benign APKs are not provided instead, the SHA-256 of the benign APKs are provided therefore using Androzoo [18], we downloaded 55,717 benign applications listed in the Drebin benign SHA-256.

B. Feature Extraction and Selection

1) BM25 score of API and Permission Feature extraction: From the smali files obtained using decompiling each application, the API used in the application are extracted. The extracted API are assigned a global unique Identifier,

which is similar across all the applications. The frequency of each unique API used in an application is extracted. Using equation 1 the BM25 score of each API is calculated with respect to each of the applications. BM25 score of each API is influenced by three factors, i.e., the frequency of the API, the Inverse Document frequency of the API, and the length of the application. The frequency of the API is useful in deciding if an API is relevant with respect to an application as example SMS (Short Message Service) malware application will have a high frequency of SMS-related APIs. The inverse document frequency is used to scale down the score of commonly occurring APIs among all the applications. The JAVA API String is a common API that is used in almost every application, therefore, it has less significance with respect to the application. Since larger application with lots of functionality are expected to contain a higher number of APIs and the frequency of the APIs is likely to be must higher than smaller applications the BM25 score is therefore influenced by the length of the document. More formally BM25 score of an API a_i in and application A_j is calculated using the following equation.

$$BM25(A_j, a_i) = IDF(a_i).\frac{TF(a_i, A_j).(k+1)}{TF(a_i, A_j) + k.(1 - b + b.\frac{|A_j|}{L})} \quad (1)$$

where, $TF(a_i, A_j)$ is the count of the number of times API a_i appear in app A_j. $|A_j|$ is the length of the application, L is the average app length of the dataset. k is set to 2.0 and b is set to 0.75. $IDF(a_i)$ is the Inverse Document Frequency of an API $a(i)$ given by equation 2 where, N is the total number of application $n(a_i)$ is the number of application that contain API a_i.

$$IDF(a_i) = ln\left(\frac{N - a_i + 0.5}{n(a_i) + 0.5} + 1\right) \quad (2)$$

To access the functionality of the device or the functionality of another application, the developer needs to declare the permission needed in the manifest file so that the user is asked for the requested permission when the apps are executed. To request an action from app components such as Activities, Services, Broadcast Receiver etc. Intents are used. The Intents used in an application are declared in the manifest file from where the permission and intents used by an application are extracted. A binary vector is constructed for each application where if a particular permission or intent is used by the application the corresponding entry is set to 1 otherwise, 0.

2) Feature Selection: The number of unique API's extracted is huge. As an example, in the data set CICMaldroid2020 more than 1.7 million unique APIs are extracted. To reduce the computation time and increase the efficiency of the classification, we select 1000 most important API. To select the API, linear regression is trained to classify the application into Malware and Benign application using BM25 score of API as a feature, using Sklearn feature selection API selectFromModel we extract the top 1000 APIs based on its importance weight. After the feature selection, each application A_j will have 1000 API $a_{1...1000}$ with corresponding

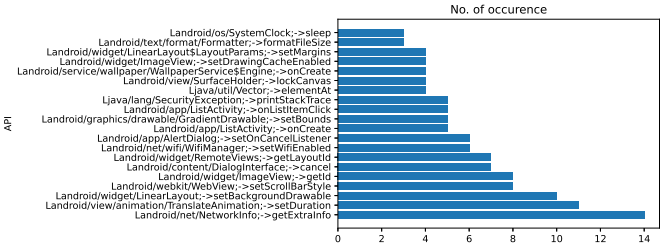

Fig. 3. Frequency of APIs unique to adware when comparing the 50 most common APIs in adware and benign application

BM25 score assigned to each API. The vector of BM25 score of length 1000 for each application along with the permissions and intent binary vector of 1's and 0's is use as a feature for classification.

3) Importance of BM25 score of API: To find out how the BM25 score of API helps in the classification, we took a sample of 50 adware malware applications and 50 Benign applications. For each application in the 50 adware application, we extracted the top 10 API with the highest BM25 score similarly, the top 10 API with the highest BM25 score is extracted for each benign application. To find the difference between the top APIs between adware and benign application, API's that occur in both benign and adware application is removed. After the removal, a frequency count of the Top APIs extracted from the adware application is performed. The frequency plot of the APIs from adware applications that are not in benign application sample is shown in figure 3. From the frequency plot, the top API such as Network Info is commonly used to get the current state of the network and Webkitview, which is used to display webpages on top of the current screen is commonly used. This make sense since the advertisement are usually displayed while a user is online and lots of benign application will not display a webpage on top of the current application instead transfer the displaying of web pages to a web browser that is already installed on the mobile phone.

4) Importance of Permission and Intent: Any functionality access by the application should be declared inside the manifest file. Malicious apps may request permissions that are not necessary for their intended functionality or that could be used to perform harmful actions, such as accessing sensitive data or displaying unwanted advertisements. On the other hand, Malicious apps may use Intent to communicate with other apps or components in ways that are not intended or that could be used to perform harmful actions. Therefore, application functionality can be summarised by permission and Intent used. Permission and Intent used in an application are extracted from the Android Manifest file where if permission or Intent is used, the corresponding permission or Intent is set to 1; otherwise, 0. Our experiment used a total of 7869 unique Permission and Intents. Permission and Intents used in an application can differentiate between Malware and Benign applications.

C. Malware Detector

To detect the malware we experimented with different types of machine learning algorithms, namely Naive Bayes, Decision Tree, Random Forest, SVM, and Convolutional Neural Network 1D.

1) Naive Bayes: Naive Bayes is a classifier based on probability which is inspired by the Bayes rule. In the Naive Bayes algorithm, we find the maximal probability of a target class 'y' given features 'X' using:

$$\hat{y} = \underset{k \in \{1...K\}}{\operatorname{argmax}} \, p(C_k) \prod_{i=1}^{n} p(x_i|C_k) \tag{3}$$

We divide the features into two parts permission, Intents, and API BM25 score. First, using permission, intents, we estimate the probability of benign and malware for each application using a Bernoulli Naive Bayes Classifier. Later, API BM25 score is used as a feature of a Multinomial Naive Bayes classifier to estimate the probability of benign and malware for each application. The probability from both the classifier is used to train a final Multinomial Naive Bayes classifier to get the final output.

2) Decision Tree: Decision Tree is a machine learning algorithm which approximates a discrete-value target function. At the time of classification, the target function is built by asking a series of questions that lead to the greatest reduction in Gini Impurity. For a set of J classes, $i \in \{1, 2, ..., J\}$ and p_i be the fraction of the item labelled with class i in the set, then Gini Impurity of a node 'n' is calculated using:

$$I_G(n) = 1 - \sum_{i=1}^{J} (p_i)^2 \tag{4}$$

The Decision tree used permission, intents, and API BM25 score to construct the target function which is represented as a tree.

3) Random Forrest: Random Forrest is a form of bagging technique where multiple decision trees are used that are trained on a random sampling of training observations and uses random subsets of features for splitting nodes.

4) Support Vector Machine: Support Vector Machine is a machine learning algorithm that uses a decision boundary for classification. It predicts a positive class if $w^T x + b$ is positive and a negative class when $w^T x + b$ is negative. The main component of SVM is the kernel function introduced in the linear formula shown in equation 14.

$$f(x) = b + \sum_{i} \alpha_i k(x, x^i) \tag{5}$$

where, $k(x, x^i) = \phi(x).\phi(x^i)$ is the kernel that enables the algorithm to learn non-linear functions.α is a vector of coefficients which is optimized using an optimization algorithm. The algorithm is trained by using permission, intents, and API BM25 score as a feature.

5) 1 Dimensional Convolutional Neural Network: 1D CNN is a variation of a Neural network that is used for operating on 1D data. The main building blocks include convolution, pooling, flattening, and feed-forward networks. The convolution step is shown in equation 6. A convolution operation is performed with input "x" and a kernel "h" of length k to get the output, the kernel is shifted with a stride of "s". The pooling layer is used to reduce the dimensionality of the data and to pick salient features after a convolution operation.

$$y(n) = \begin{cases} \sum_{i=1}^{k} x(n+i)h(i) & \text{if } n = 0 \\ \sum_{i=1}^{k} x(n+i+(s-1))h(i) & \text{otherwise} \end{cases} \tag{6}$$

D. Hyper Parameter Optimization and Dealing with class imbalance

Hyperparameter optimization, also known as hyperparameter tuning, is the process of choosing the optimal values for the hyperparameters of a machine learning model. In this work we used Grid Search and Particle Swarm Optimization.

1) Grid Search: Grid search is an exhaustive hyperparameter search technique where the search spans all the possible combinations of a specified hyperparameter. Grid search is quite expensive in computation since it has to evaluate all the possible combinations.

2) Particle Swarm Optimization: Particle Swarm Optimization is a population-based meta-heuristic algorithm that uses a group of particles to find an optimal solution to a problem. It involves a group of particles that move through the search space and update their positions based on their own experience and the experience of the other particles.

3) Dealing with class imbalance: The Drebin dataset class distribution is highly skewed, with 55,717 benign samples and 5,560 malware samples. This imbalance can affect the performance of the learning algorithm with a high bias towards benign samples. To compensate for the imbalance class weights are introduced to penalize the miss classification made by the malware class and at the same time reduce the class weight of the benign class by introducing the weights to the loss function. The weights are calculated using the following formula

$$w_i = \frac{m}{n * s_i} \tag{7}$$

where w_i is the weight of class i, m is the number of samples in the dataset, n is the number of class and s_i is the number of samples of class i in the dataset. The modified loss with the weight introduced is given by the following equation.

$$L = \frac{1}{N} \sum_{i=1}^{N} \left[-(w_0(y_i * log(\mathrm{y}_i^{'})) + w_1((1 - y_i) * log(1 - \mathrm{y}_i^{'}))) \right] \tag{8}$$

where, w_0 is the weight of class 0, w_1 is the weight of class 1 y_i is the true label,$y_i^{'}$ is the predicted label.

TABLE I
PERFORMANCE RESULT WITHOUT PSO (IN %)

Model	**Features**	**Drebin**				**CICMalDroid**			
		Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
NB	PI	92.75	58.65	69.18	63.48	95.53	96.26	97.96	97.1
DT	PI	96.7	76.09	91.79	83.2	96.47	98.03	97.34	97.68
RF	PI	97.94	86.29	91.09	88.62	97.53	98.39	98.39	98.39
SVM	PI	98.22	93.61	85.76	89.51	97.1	98.17	98.04	98.1
CNN	PI	97.81	88.9	87.67	88.28	96.45	97.93	97.43	97.68
NB	PI+BM25	93.35	60.66	75.58	67.3	93.42	60.15	77.91	67.88
DT	PI+BM25	97.76	86.56	89.11	87.82	89.58	91.03	95.82	93.36
RF	PI+BM25	98.96	98.41	89.98	94.01	95.77	97.21	97.26	97.24
SVM	PI+BM25	96.95	98.19	67.95	80.32	**99.04**	**99.32**	**99.43**	**99.37**
CNN	PI+BM25	**98.13**	**99.61**	**94.09**	**95.33**	98.7	99.12	99.27	99.16

TABLE II
PERFORMANCE RESULT USING PSO (IN %)

Model	**Features**	**Drebin**				**CICMalDroid**			
		Accuracy	Precision	Recall	F1	Accuracy	Precision	Recall	F1
NB	PI+BM25	93.73	60.33	85.29	70.67	96.99	99.58	96.46	98
DT	PI+BM25	78.52	26.26	76.95	39.16	95.39	96.41	97.61	97
RF	PI+BM25	99.05	97.09	92.3	94.63	95.72	98.83	95.54	97.15
SVM	PI+BM25	98.18	86.26	94.43	90.16	**99.08**	**99.20**	**99.6**	**99.4**
CNN	PI+BM25	**98.37**	**97.83**	**94.83**	**96.31**	98.7	99.35	98.95	99.15

TABLE III
RESULT COMPARISON WITH OTHER WORKS (IN %)

Author	Dataset	Benign/Malware apks	Accuracy	Precision	Recall	F1-Score
Rana et al.(2018) [19]	Drebin	5,560/5,560	94.33	95.00	94.00	94.00
Rana et al.(2018) [20]	Drebin	5,560/5,560	97.24	97.58	96.88	97.23
Zhang et al.(2019) [22]	Drebin	5,560 (only Malware)	93.90	-	94.00	-
Bai et al.(2020) [21]	Drebin	5,900/5,560	96.00	97.00	95.00	96.00
Our work	Drebin	55,717/5,560	**98.37**	**97.83**	94.83	96.31
Mahdavifar (2020) [16]	CICMaldroid	1479/11,598	96.70	99.16	96.54	97.84
Our work	CICMaldroid	3,696/12,152	**99.08**	**99.20**	**99.6**	**99.4**

TABLE IV
BEST HYPERPARAMETER FOUND USING PSO

Model	**Dataset**	
	Drebin	CICMalDroid
NB	var smoothing=0.18	var smoothing=0.17
DT	spliter=best, max_depth=10, min_samples_leaf=14, min_weight_ fraction_leaf=0.042, max_features='sqrt', max_leaf_nodes=78	spliter=random, max_depth=13, min_samples_leaf=9, min_weight_fraction- _leaf=0.21, max_features='sqrt', max_leaf_nodes=37
RF	n_estimators=704, max_depth=22, min_samples_split=3, min_samples_leaf=6	n_estimators=401, max_depth=19, min_samples_split=8, min_samples_leaf=4
SVM	C=11.27, gamma=0, kernel=linear	C=41.30, gamma=0.39, kernel=polynomial
CNN	no.of_layer=8, embedding_size= 500, lr=0.04,beta1=0.84, beta2=0.70, epsilon=0.71, convolution_size=106	no.of_layer=6, embed- ding_size=422,lr=0.04, beta1=0.23, beta2=0.30, epsilon=0.14, convolution_size=124

V. EXPERIMENTAL PERFORMANCE EVALUATION

1) Impact of Feature Selection on the feature size: Using Sklearn feature selection API SelectFromModel we select 1000 APIs from more than 1.7 million API's based on their importance weight. It is found that selecting more than 1000 does not increase the accuracy of the classification algorithm, therefore it is decided to select only the top 1000 most important APIs. The feature selection significantly reduces the number of APIs required for classification and thus improves the efficiency of classification.

2) Impact of BM25 score on the classification: One of the main benefits of using the BM25 score for Android malware classification is that it can effectively weigh the importance of the APIs based on their rarity and relevance, which can improve the discriminative power of the model. The main parameter of the BM25 score are the frequency of the API, the Inverse Document frequency of the API, and the length of the application. Taking the example of adware application as shown in figure 3, the API with the highest frequency is NetworkInfo class calling getExtrainfo method among the APIs unique to adware when comparing the 10 most common APIs in adware and benign applications. This leads to a higher term frequency of API NetworkInfo class calling getExtraInfo in

the Adware application in comparison to Benign Application. Additionally, since NetworkInfo class calling getExtrainfo is not a common API the Inverse Document Frequency will be high as compared to commonly occuring API such as Button class performClick method which is used to performed all normal actions associated with clicking a button. Therefore, it can be safely assumed that API such as NetworkInfo class calling getExtrainfo will have higher BM25 score as compared to Benign Applications. The difference in BM25 API Score in Malware application and Benign application thus helped in the overall classification of Malware and Benign.

3) Impact of optimization methods on the ML models: From the result shown in table I and II, it can be seen that using PSO slightly improve the overall accuracy of all the algorithm. PSO hyperparameter optimization can go wrong if the search space initialized is not good, which can be seen in the case of DT in table II. Although PSO is a good hyperparameter search algorithm, if the search space is not carefully selected, it may not converse. Grid search can also achieve good accuracy even though the computational cost is high since every combination of the hyperparameter should be tested. As compared to grid search, PSO is a faster algorithm for hyper-parameter search since it searches the hyperparameter using a population of particles which leverages the local and global knowledge of the population. The best hyperparameters found for the machine learning and deep learning algorithm are shown in table IV.

4) Overall Analysis: In the case of Drebin dataset CNN train with Permission, Intent and BM25 score of API performs best as compared to other algorithm whereas in the case of CICMalDroid SVM outperforms all the other algorithms. The difference in the best classifier can be attributed to the fact that deep learning algorithms outperform traditional machine learning algorithms when the number of samples becomes large. Since Drebin is a significantly larger dataset, CNN does better as compared to other algorithms. The comparison with other works is shown in table III, which shows that the proposed method is comparable to the state-of-the-art methods.

VI. Conclusion

The research on Android malware classification using the BM25 score of APIs and Permission and Intent features shows promise to improve the accuracy of malware classifiers. The BM25 score takes into account the frequency and position of the API in the app, as well as the length of the app and the overall frequency of the API in the dataset. The use of the BM25 score, which assigns importance scores to APIs based on their functionality, helped to improve the performance of the classifier. In addition, the combination of BM25 score with Permission and Intent features along with PSO optimisation resulted in improved accuracies on the benchmark datasets.

References

[1] Threatfabric, Deceive the Heavens to Cross the sea, Threatfabric.com, 2021. [Online]. Available: https://www.threatfabric.com/blogs/deceive-the-heavens-to-cross-the-sea.html. [Accessed: 13- Nov- 2022]

[2] Mcafee, Technical analysis of BRATA, Mcafee.com, 2021. [Online]. Available: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-brata.pdf. [Accessed: 30- Oct- 2022]

[3] Su, X., Zhang, D., Li, W. & Zhao, K. A deep learning approach to android malware feature learning and detection. *2016 IEEE Trustcom/BigDataSE/ISPA*. pp. 244-251 (2016)

[4] Su, X., Shi, W., Qu, X., Zheng, Y. & Liu, X. DroidDeep: using Deep Belief Network to characterize and detect android malware. *Soft Computing*. pp. 1-14 (2020)

[5] Yuan, Z., Lu, Y., Wang, Z. & Xue, Y. Droid-sec: deep learning in android malware detection. *Proceedings Of The 2014 ACM Conference On SIGCOMM*. pp. 371-372 (2014)

[6] Alotaibi, A. Identifying Malicious Software Using Deep Residual Long-Short Term Memory. *IEEE Access*. **7** pp. 163128-163137 (2019)

[7] Wang, W., Zhao, M. & Wang, J. Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. *Journal Of Ambient Intelligence And Humanized Computing*. **10**, 3035-3043 (2019)

[8] Nix, R. & Zhang, J. Classification of Android apps and malware using deep neural networks. *2017 International Joint Conference On Neural Networks (IJCNN)*. pp. 1871-1878 (2017)

[9] Vinayakumar, R., Soman, K. & Poornachandran, P. Deep android malware detection and classification. *2017 International Conf on Advances in Computing, Communications & Informatics*. pp. 1677-1683 (2017)

[10] Ryszard Wiśniewski, C. Apktool. (https://ibotpeaches.github.io/Apktool/, 2021), Online; accessed 20 January 2022

[11] Hou, S., Ye, Y., Song, Y. & Abdulhayoglu, M. Hindroid: An intelligent android malware detection system based on structured heterogeneous information network. *Proceedings Of The 23rd ACM SIGKDD International Conference On Knowledge Discovery And Data Mining*. pp. 1507-1515 (2017)

[12] Pektaş, A. & Acarman, T. Learning to detect Android malware via opcode sequences. *Neurocomputing*. (2019)

[13] Yuan, Z., Lu, Y. & Xue, Y. Droiddetector: android malware characterization and detection using deep learning. *Tsinghua Science And Technology*. **21**, 114-123 (2016)

[14] Alzaylaee, M., Yerima, S. & Sezer, S. DL-Droid: Deep learning based android malware detection using real devices. *Computers & Security*. **89** pp. 101663 (2020)

[15] Bibi, I., Akhunzada, A., Malik, J., Ahmed, G. & Raza, M. An Effective Android Ransomware Detection Through Multi-Factor Feature Filtration and Recurrent Neural Network. *2019 UK/China Emerging Technologies (UCET)*. pp. 1-4 (2019)

[16] Mahdavifar, S., Kadir, A., Fatemi, R., Alhadidi, D. & Ghorbani, A. Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning. *2020 IEEE Intl Conf On Dependable, Autonomic And Secure Computing, Intl Conf On Pervasive Intelligence And Computing, Intl Conf On Cloud And Big Data Computing, Intl Conf On Cyber Science And Technology Congress*. pp. 515-522 (2020)

[17] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K. & Siemens, C. Drebin: Effective and explainable detection of android malware in your pocket.. *Ndss*. **14** pp. 23-26 (2014)

[18] Allix, K., Bissyandé, T., Klein, J. & Le Traon, Y. Androzoo: Collecting millions of android apps for the research community. *2016 IEEE/ACM 13th Working Conference On Mining Software Repositories (MSR)*. pp. 468-471 (2016)

[19] Rana, M., Gudla, C. & Sung, A. Evaluating machine learning models for android malware detection: a comparison study. *Proceedings Of The 2018 VII International Conference On Network, Communication And Computing*. pp. 17-21 (2018)

[20] Rana, M., Rahman, S. & Sung, A. Evaluation of tree based machine learning classifiers for android malware detection. *International Conference On Computational Collective Intelligence*. pp. 377-385 (2018)

[21] Bai, H., Xie, N., Di, X. & Ye, Q. FAMD: A Fast Multifeature Android Malware Detection Framework, Design, and Implementation. *IEEE Access*. **8** pp. 194729-194740 (2020)

[22] Zhang, H., Luo, S., Zhang, Y. & Pan, L. An efficient Android malware detection system based on method-level behavioral semantic analysis. *IEEE Access*. **7** pp. 69246-69256 (2019)