Applied Soft Computing Journal 102 (2021) 107069

Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

Deep learning feature exploration for Android malware detection

Nan Zhang [a,b], Yu-an Tan [a], Chen Yang [a], Yuanzhang Li [a,*]

[a] School of Computer Science, Beijing Institute of Technology, Beijing, 100081, China
[b] Institute of Artificial Intelligence and Blockchain, Guangzhou University, Guangzhou, 510006, China

ARTICLE INFO

Article history:
Received 26 July 2020
Received in revised form 11 November 2020
Accepted 22 December 2020
Available online 2 January 2021

Keywords:
Malware detection
Android security
Text classification
Deep learning
Smart city

ABSTRACT

Android mobile devices and applications are widely deployed and used in industry and smart city. Malware detection is one of the most powerful and effective approaches to guarantee security of Android systems, especially for industrial platform and smart city. Recently, researches using machine learning-based techniques for Android malware detection increased rapidly. Nevertheless, most of the appeared approaches have to perform feature analysis and selection, so-called feature engineering, which is time-consuming and relies on artificial experience. To solve the inefficiency problem of feature engineering, we propose TC-Droid, an automatic framework for Android malware detection based on text classification method. The core idea of TC-Droid is derived from the field of text classification. TC-Droid feeds on the text sequence of APPs analysis reports generated by AndroPyTool, applies a convolutional neural network (CNN) to explore significant information (or knowledge) under original report text, instead of manual feature engineering. In an evaluation with different number of real-world samples, TC-Droid outperforms state-of-the-art model (Drebin) and several classic models (NB, LR, KNN, RF) as well. With multiple experimental settings and corresponding comparisons, TC-Droid achieves effective and flexible performance in Android malware detection task.

1. Introduction

Smart city technology and Android OS were widely deployed on various information fields in recent years, such as smart government, intelligent transportation, energy & resource management, etc. [1–4]. With the great convenience supplied by Android platform, the malware targeting Android also causes critical threats that lead to financial loss, data leak, national security concerns and terroristic events [5–8].

On account of the excellent openness and flexibility offered by Android, it has captured more than 80% of smartphone users. On another hand, Android has also become a major victim of malware [9]. According to the report of GDATA [10], about 4 million new malicious APPs were born in 2019. On average, hackers create an infected APP every eight seconds. The rapidly increasing of malware brings much threat to cyber-security. Well-performed detection methods (or frameworks) for Android malware are urgently required to enhance mobile security [11]. Mobile security researchers have proposed various defense approaches to mitigate the risk of malicious APPs, especially the malware detection methods based on machine learning techniques that seem to be a panacea. With the application of deep learning, this magical technology also attracts the attention of many security researchers [12–14]. According to the underlying feature, the state-of-the-art malware detection methods using machine learning techniques can be generally divided into three categories: *static feature* [15–18], *dynamic feature* [19,20], and *mixed feature* [21,22].

In the most of previous work, feature selection is a necessary process for training machine learning models [23–26]. To improve model performance, researchers apply feature selection to reduce redundant or irrelevant features, refine significant features. Nevertheless, the processing of feature trimming usually breaks the integrity of the underlying data, a time-consuming and experiential task besides. In this paper, we propose TC-Droid, a fortifying Android malware detection framework using text classification method and feeding on static analysis reports of APKs, in which feature selection is unnecessary. Each analysis report contains a sequence of sentences, output by AndroPyTool which combines well-known Android APPs analysis tools such as DroidBox, FlowDroid, Strace, AndroGuard or VirusTotal analysis. The motivation of TC-Droid is to enable the model fully mine the potential meaningful features hidden in underlying data. We apply TextCNN algorithm to scan analysis reports and hope this NLP method can automatically capture deep features from the sentences. The experimental evaluation demonstrates the efficacy of our method: TC-Droid outperforms some art-of-the-state models. The contributions of this work are summarized in the following:

* Corresponding author.
E-mail address: popular@bit.edu.cn (Y. Li).

https://doi.org/10.1016/j.asoc.2020.107069
1568-4946/

- *NLP method for Android malware detection.* TC-Droid refers the text classification approach in natural language processing. We pioneeringly utilize the TextCNN algorithm to mine the text difference between malicious and benign APPs reports.
- *Non-essential feature selection.* TC-Droid takes advantage of CNN nature to automatically perform feature extracting and selecting. TC-Droid is sufficient to replace the burdensome manual feature engineering in malware detection task, and capture potential useful information in sentence sequences.
- *A prototype of TC-Droid with performance evaluation on real-world dataset Extensive experiments on real-world samples.* Extensive experiments are conducted on large-scale malware and goodware. Evaluation result shows that TC-Droid outperforms related state-of-the-art approaches.

The paper is organized as follows. First, background and related work is given in Section 2. The proposed framework with its preliminaries and methodology is explained in Section 3. The performance of experiments and their discussion is given in Section 4, where we use the text classification method as a baseline. Lastly, we conclude & discuss future works in Section 5.

2. Background & related works

In this section we first describe the three different methods using machine learning for Android malware detection, followed by recent state-of-the-art research works using deep learning.

2.1. ML-based Android malware detection methods

2.1.1. Static analysis

The authors of ANASTASIA [27] proposed a system that uses several classical machines learning classifiers and deep neural networks for Android malware detection. They designed and built a static analysis tool to extract some sensitive features. Experiments on large-scale and well-labeled dataset showed a 97.3% true positive rate. DroidAPIMiner [17] extracts the most sensitive 169 package-leveled API calls, that are commonly found in malicious apps (as compared to benign ones). It also extracts other package-level and parameter information as features from a large set of Android malware. the work reports a 99% accuracy with a false positive rate of 2.2%. Drebin [28] built eight feature sets by extracting features from the manifest file and the source code of apps. It trains a Support Vector Machine (SVM) classifier to classify these features for malware detection and achieves relatively good performance in the detection of malware families. DroidSIFT [29] is a novel system that uses semantic features extracted from weighted contextual API dependency graphs to classify Android malware. The system can effectively fight against malware variants and zero-day malware through graph similarity metrics. MAMADROID [30] utilizes a Markov chain to present the sequences of API calls and extracted features, while the classification of behavior model is performed using different classification algorithms. DroidMat [31] extracts features, such as permissions and API calls, from the Android manifest file, after which the K-nearest neighbor algorithm is used to classify benign apps and malware. The classifier can be further improved by using the K-means algorithm.

2.1.2. Dynamic analysis

DroidScope [32] constructs OS-level and Java-level semantic views for dynamic android malware analysis. It observes changes in the environment (i.e threads and processes), system calls, and Dalvik instructions for dynamic analysis. Endroid [26] automatically extracts various kinds of dynamic behavior features and then use a feature selection algorithm to determine risky behavior. It gives excellent performance in Android malware detection when applying ensemble learning stacking. MADAM [33] defines the different malware types by monitoring features belonging to four different Android levels. It present, a novel host-based malware detection system for Android devices, which identifies the malware based on misbehavior classes.

2.1.3. Hybrid analysis

StormDroid [21] is based on the combination of static and dynamic feature extraction. It is the first to develop a real-time analysis system based on streaming machine learning. MARVIN [22] is a system that uses malicious scores and applies machine learning techniques to determine unknown Android applications. Its evaluation is based on a hybrid analysis on a well-labeled dataset and shows 98.24% accuracy. Mobile-Sandbox [34] is a system designed to automatically implement hybrid analysis on Android applications with machine learning. Work in [35] attempted to reveal the difference between benign and malicious apps in terms of their run-time behaviors over time.

2.2. Android malware detection using deep learning

Droid-sec [36] is a DL-based method that uses hybrid analysis to detect Android malware. Deep Belief Network (DBN) models are utilized to classify malware from benign ones and the model obtained 96% accuracy. DroidDeep [37] is a deep learning method for Android malware detection. It first extracts five main kinds of static features from 3,986 benign and malware apps (total of 32,247 features). Then it uses the DBN model to learn and finally classify by using an SVM. Results show that DroidDeep achieves 99.4% accuracy. A new version of DroidDeep was proposed in [38], which provides a DBN-based model to learn unique behavioral characteristics of Android apps and thus improving detection performance. DeepRefiner [12] is a semantic-based deep learning system, which uses Long Short Term Memory (LSTM) with multiple hidden layers to automatically extract semantic information from the method-level bytecode of Android apps. The results show that the accuracy of DeepRefiner is up to 97.74%.

A deep learning system using conventional neural networks for malware detection in Android is presented in [13]. It uses raw opcode sequences which are extracted from apps as features. A large-scale dataset evaluation shows the reasonable performance of this technique. The work in [39] presents the first multi-modal deep learning method for Android malware detection. They provided two novel feature representation as existence-based and similarity-based feature extraction methods. The work in [14] proposes an anti-malware system that uses customized deep learning architectures that can detect and attribute the Android malware via opcodes extracted from application byte-code.

DroidDetector [40] is an online, automatic DL-based Android malware detection engine. It emphasizes that deep learning is a good method in classifying malware based on a large number of samples. It also claims to have 96.76% detection accuracy under different conditions, which is much higher than most traditional machine learning techniques. Work in [41] proposed a novel Android malware detection method that can identify malware using the different URLs visited by apps. A multi-view neural network is used to create multiple segments and generate multiple views of the inputs. Rich semantic information is preserved from inputs for classification without complex feature engineering.

Most of the existing works use manual feature selection, namely selecting top features. In contrast, our work automatically finds more potential but sensitive features using text classification method in the field of Natural Language Processing (NLP) What is more, to the best of our knowledge, there is only one related work which attempted to apply text classification method to Android malware detection [42].

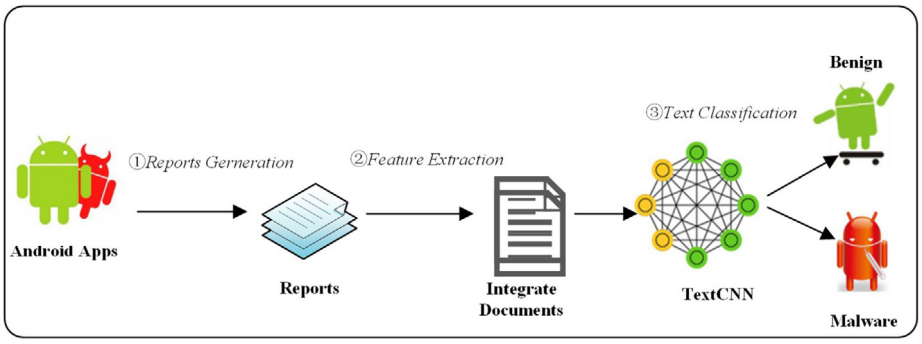

Fig. 1. Framework of *TC-Droid* for Android malware detection.

3. Proposed methodology

In this section, we describe TC-Droid, a novel framework that uses a text classification method to perform the Android malware detection task. First, we describe the preliminary information on the Android application structure and the text classifies TextCNN, for a better understanding of different components described later. The abstract system model is shown in Fig. 1. The three main components are report generation, feature extraction, and text classification, which are also described in detail below.

3.1. Preliminaries

It is important to note the basic structure of Android applications and the TextCNN model.

3.1.1. Android application structure

The application programming kit (apk) files can be considered similar to a compressed (zip) or archived file, that contains several important resource files. The five main components of an apk file are; `METF — INF`, `res`, `Classes.dex`, `resources.arsc`, and `AndroidManifest.xml`.

- `AndroidManifest.xml`: This is an XML file, that stores the meta-data such as package name, permissions required, definitions of one or more components (i.e. activities, services, broadcast receivers, or content providers), min and max version support, and linked libraries, etc.
- `Classes.dex`: This is an executable file that stores the Dalvik bytecode that can be executed on the Dalvik Virtual Machine (DVM).

3.1.2. Text classification using TextCNN

Convolutional neural networks have been proven to work very well for text classification. We use the TextCNN as presented in [43] to complete the text classification task. The model used in the paper mainly includes five layers, that are (1) embedding layer, (2) convolutional layer, (3) max-pooling layer, (4) fully connected layer, and (5) softmax layer. In the model, each sequence is a k-dimensional vector (padded where necessary), and each input is present as a series of n sequences; thus each input is a feature map of dimensions of $n \times k$. Following this, we apply a max-over-time pooling operation on the feature map. The features which correspond to the filter with the highest value (max value) are taken. The objective is to capture the most important features, which are those that have the highest value for each feature map.

3.2. Reports of generation process

It can be observed from Fig. 1, that the first step is to generate the reports. In this step, we use first use reverse engineering to analyze the *apk* files and then generate the reports. All reports are in a JavaScript Object Notation (JSON) format. A report contains four parts: *Dynamic_analysis*, *Pre_static_analysis*, *Static_analysis*, *VirusTotal*,[1] as shown in Fig. 2. In our approach, we do not consider *Dynamic_analysis*.

3.3. Feature extraction process

To detect Android malware using a text classification method, our framework requires more meaningful content to describe each app. To this end, we mainly extract four types of static features from reports: **Permission, Service, Intent, Receiver**. Each of these is explained below.

- **Permission:** This is one of the most important security mechanisms in an Android system. It provides authority to an application to get access to sensitive resources or execute some high-risk operations.
- **Service:** This is a list of different service points used by the app. This list of service names is an important feature set, as it may help in identifying well-known components of different malware.
- **Intent:** Similar to services, intents are also a component of the Android system. Besides, they are a key communication element as they enable an app to seek permission from the OS to execute other application components or processes. For example, an intent can require the execution of actions such as enabling camera or accessing photo album, etc.
- **Receiver:** This a global listener, which belongs to one of the four major components of Android. The broadcast receiver is used to receive broadcast intents asynchronously.

3.4. Text classification process

Once the above-mentioned processes are completed, we train the TextCNN for text classification. Using this model, we can capture the key information of the classification from the features. The input of the model is two feature documents (i.e., benign and malicious), as shown in Fig. 3.

4. Implementation & experimental evaluation

To evaluate the working performance of the proposed TC-Droid scheme, we have conducted extensive experimentation. The experiments are conducted using a native Ubuntu 18.04

[1] https://www.virustotal.com/.

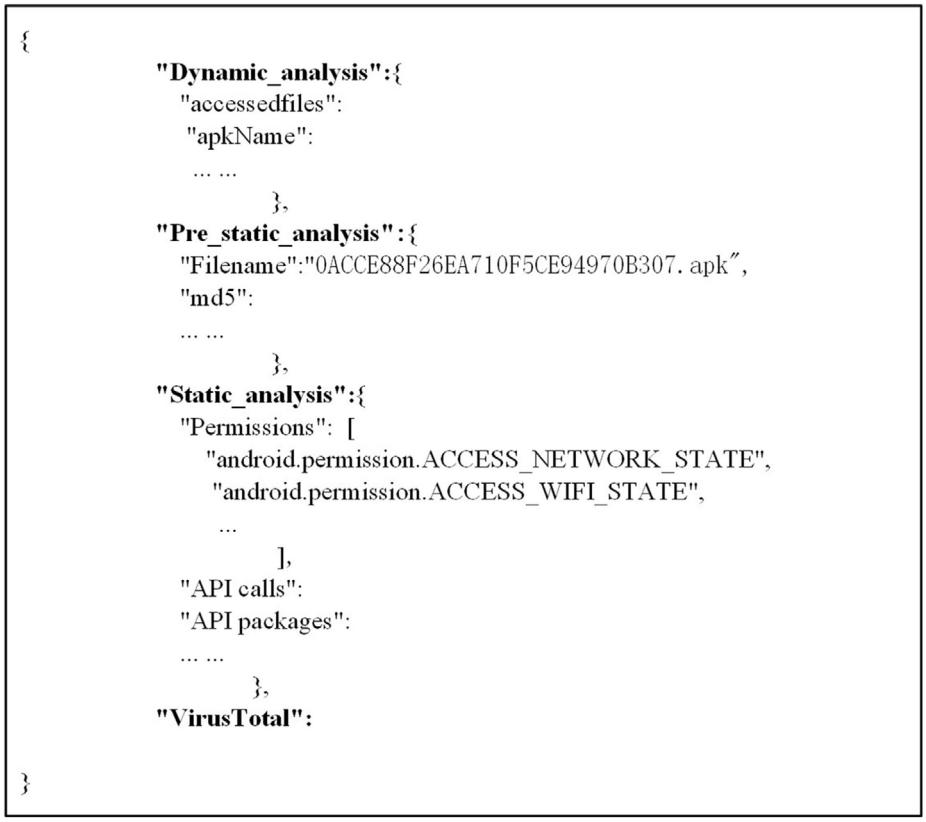

Fig. 2. Example of JSON report with four different parts.

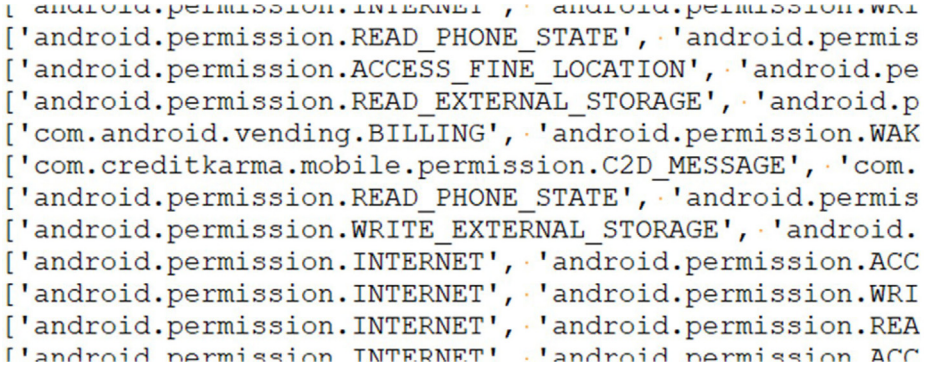

Fig. 3. An example of feature document.

platform on an Intel Xeon E5-2620 v4 CPU at 2.10 GHz with 32 GB RAM and dedicated Nvidia GeForce RTX 2080 Ti GPU. In the experiments, the GPU is utilized to accelerate the convolutional neural network algorithm. The proposed framework is implemented using Python and compiled in the Keras scientific computing environment. The network parameters have been optimized using the Adadelta system, during the training process. The learning rate is 1e−2 for 10 epochs with a batch size of 16. The performance has been evaluated using the following three experiments.

Detection performance: In the first experiment, we evaluate the detection performance of the proposed TC-Droid. The objective is to determine the time it takes to identify a malicious app.

Run-time performance: In this experiment, we evaluate the run-time performance of the complete system.

Table 1
Datasets used in experiments.

Number of Samples	D1	D2	D3	D4
(Goodware,Malware)	(1270,1176)	(2515,2517)	(3469,3219)	(5000,5000)

Results comparison: Finally, in the last set of experiments, we compare the detection results of the TC-Droid approach with a feature selection based model.

4.1. Datasets

To evaluate our proposed method, we consider four datasets with a different number of real-world apps as shown in Table 1. Each data set has a large number of goodware and malware as shown here. It is worth mentioning that the samples of the dataset D4 are significantly larger than the others.

The benign applications are crawled from Anzhi, a popular third-party Android app market in China. The malware samples are from the Contagio Community[2] and Android Malware Genome Project [44], which includes different types of Android malware such as Trojans, Backdoors, Ransomware, and Virus, etc. We then clean the dataset by removing replicated or invalid applications by using the AndroGuard tool.[3] Besides, all applications are analyzed with the online malware detection engine VirusTotal. We flag an application as benign only if the vast majority of malware detectors detect it as such, otherwise, it is malicious. All samples were analyzed by using AndroPyTool.[4]

4.2. Evaluation parameters

The performance of the malware detection approach is measured using the following: Accuracy (ACC), Precision, Recall, and F1-score. It is important to note that, in this evaluation section *true positive* (TP) indicates that malware samples were correctly classified as malware, while truly classified benign reports are labeled as *true negative* (TN). *False positive* (FP) indicates the misclassification of a benign report as malware, while malware instances classified as benign are labeled as *false negative* (FN). Each of the parameters is computed using the following equations.

Accuracy is defined as the ratio of correctly predicted samples to the total count of all samples, and is calculated as,

$$ACC = \frac{TP + TN}{(TP + FP + TN + FN)}. \quad (1)$$

Precision is defined as the ratio of correctly predicted positive samples to the number of all predicted positive samples, and is computed as,

$$Precision = \frac{TP}{(TP + FP)}. \quad (2)$$

Recall is defined as the ratio of correctly predicted positive samples to the total number of true positive samples, and determined by,

$$Recall = \frac{TP}{(TP + FN)}. \quad (3)$$

F1-score is computed as,

$$\text{F1-score} = 2 \times \frac{Precision \times Recall}{(Precision + Recall)}. \quad (4)$$

4.3. Detection performance

In this experiment, to determine the detection performance of the TC-Droid, we evaluate our framework under three datasets D1, D2, D3. Each dataset is split into 90% for training and 10% for testing. we use AndroPyTool to generate the apps' reports. We compare the performance of our proposed system with our four common machine learning methods K-nearest neighbor (KNN), Random Forest (RF), Logistic Regression (LR), Naive Bayes (NB) with TF–IDF algorithm and a text classification model FastText [45]. The results are displayed in Table 2.

Table 2 shows that the proposed framework TC-Droid exceeds the common machine learning methods in most of the performance metrics. TC-Droid achieves the highest accuracy rates of 96.72% and precision of 97.60% under dataset D1. Besides, the performance of our system is stable under different number of samples.

Table 2
Detection performance of three datasets.

Dataset	Metric	FastText	NB	LR	KNN	RF	TC-Droid
D1	ACC	0.9426	0.9426	0.9344	0.9098	0.9508	**0.9672**
	Precision	0.9065	0.9658	0.9688	0.9545	0.9748	**0.9760**
	Recall	**0.9921**	0.9187	0.9118	0.8607	0.9280	0.9606
	F1-score	0.9422	0.9426	0.9334	0.9096	0.9508	**0.9671**
D2	ACC	0.9681	0.9284	0.9463	0.9483	0.9662	**0.9682**
	Precision	0.9644	0.9522	0.9676	**0.9871**	0.9760	0.9644
	Recall	**0.9721**	0.9087	0.9264	0.9090	0.9568	**0.9721**
	F1-score	**0.9681**	0.9284	0.9463	0.9482	0.9662	**0.9681**
D3	ACC	0.9611	0.9341	0.9476	0.9371	0.9580	**0.9611**
	Precision	0.9486	0.9450	0.9633	0.9801	0.9634	0.9511
	Recall	0.9765	0.9154	0.9320	0.8919	0.9432	0.9734
	F1-score	0.9610	0.9339	0.9476	0.9369	0.9579	**0.9610**

Table 3
Comparison of detection performance.

Approach	ACC	Precision	F1-score	Recall
TC-Droid	**0.966**	0.946	0.966	0.984
Drebin	0.952	0.955	0.952	0.949

4.4. Run-time performance

In this section, we further evaluate the computation overhead of TC-Droid. Fig. 4 shows the time cost for feature extraction and the time cost for model training under different datasets.

It can be observed that the processing time increases smoothly with the number of increasing samples. Given this, we can extract features from more than 6000 reports in approximately 23 s, and the time cost for model training is less than the 170 s. Based on these numbers, we believe, that the performance of TC-Droid is acceptable for the real-world and it can be used for analyzing a larger number of samples for Android malware detection purposes.

4.5. Detection performance comparison with existing methods

To evaluate the performance of TC-Droid against other detection systems, we investigate other similar approaches from the literature. In particular, we consider Drebin [28]. Drebin performs a broad static analysis, building as many as features from the manifest file and the source code of apps as possible, such as used permissions, API calss and intents . It trains a SVM classifier to classify these features for malware detection and achieves relatively good performance. First, we randomly select samples (D4) as the training set (90%) and as the testing set (10%). Then, we use AndroPyTool to generate the apps' reports. Finally, we compare detection results in the same dataset. In this section, we extract more potential features from reports, such as **Activities, API_packages, System_commands, Opcodes**, which helps improving performance of TC-Droid.

As shown in Table 3, we can see that TC-Droid performance better than Drebin among almost the four experiment metrics. In particularly, TC-Droid can achieve 96.6% accuracy which is higher than Drebin (95.2%). The results clearly illustrate that TC-Droid is able to detect malware with better performance than the existing approaches that are based on traditional machine learning algorithms.

Our work innovatively removes the traditional time-consuming feature selection process and uses TextCNN to discover potential features. The experiments prove that our proposed method is effective, and also shows that artificial feature selection is sometimes not correct.

[2] http://contagiominidump.blogspot.com/.
[3] http://code.google.com/p/androguard/wiki/DatabaseAndroidMalwares.
[4] https://github.com/alexMyG/AndroPyTool/.

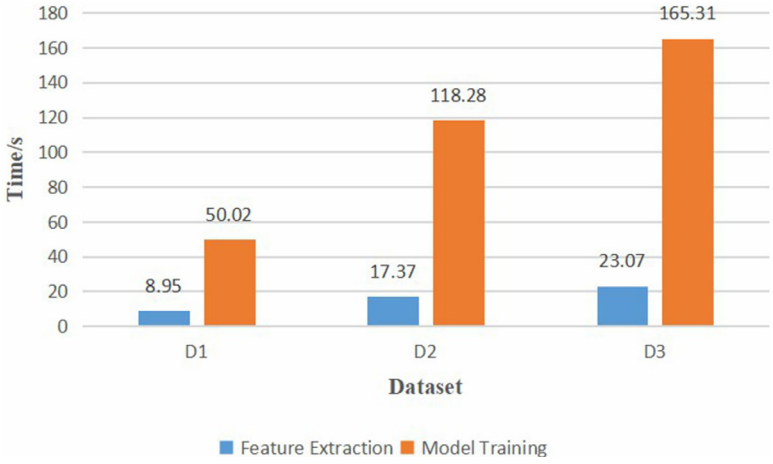

Fig. 4. Time cost for feature extraction and model training.

5. Conclusions and future work

To improve the detection of malware for Android-based systems in smart cities, we propose a novel Android malware framework using text classification with a convolutional neural network. We proposed TC-Droid, does not require hand-engineered feature selection in the domain of Android malware detection. The methodology and experiments show that by doing so better results can be obtained. It is more efficient and retains a higher detection accuracy. This is mainly due to the use of TextCNN to auto-detect feature representations (word sequences of reports). In the future, the approach will be extended to both dynamic and static features. It is worth mentioning that dynamic features could be extracted in real devices.

CRediT authorship contribution statement

Nan Zhang: Conceptualization, Methodology, Software. **Yu-an Tan:** Visualization, Investigation. **Chen Yang:** Software, Validation. **Yuanzhang Li:** Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China under Grant 61876019; and Zhejiang Lab, China (NO. 2020LE0AB02).

References

[1] M. Daraghmeh, I.A. Ridhawi, M. Aloqaily, Y. Jararweh, A. Agarwal, A power management approach to reduce energy consumption for edge computing servers, in: 2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC), 2019, pp. 259–264.
[2] F. Al-turjman, H. Zahmatkesh, L. Mostarda, Quantifying uncertainty in internet of medical things and big-data services using intelligence and deep learning, IEEE Access 7 (2019) 115749–115759.
[3] B.D. Deebak, F.M. Al-Turjman, M. Aloqaily, O. Alfandi, An authentic-based privacy preservation protocol for smart e-healthcare systems in IoT, IEEE Access 7 (2019) 135632–135649.
[4] G. Srivastava, R.M. Parizi, A. Dehghantanha, The future of blockchain technology in healthcare internet of things security, 2020.
[5] W. Yaokumah, M. Rajarajan, J.-D. Abdulai, I. Wiafe, F. Katsriku, Modern Theories and Practices for Cyber Ethics and Security Compliance, 2020, http://dx.doi.org/10.4018/978-1-7998-3149-5.
[6] M. Nassiri, H. HaddadPajouh, A. Dehghantanha, H. Karimipour, R.M. Parizi, G. Srivastava, Malware elimination impact on dynamic analysis: An experimental machine learning approach, in: Handbook of Big Data Privacy, 2020.
[7] F. Ullah, H. Naeem, S. Jabbar, S. Khalid, M.A. Latif, F. Al-turjman, L. Mostarda, Cyber security threats detection in internet of things using deep learning approach, IEEE Access 7 (2019) 124379–124389.
[8] A. Yeboah-Ofori, J.-D. Abdulai, F. Katsriku, Cybercrime and risks for cyber physical systems 2019, Int. J. Cyber-Secur. Digit. Forensics 8 (1) (2019) 43–58.
[9] J.-F. Lalande, V.V.T. Tong, P. Graux, G. Hiet, W. Mazurczyk, H. Chaoui, P. Berthomé, Teaching android mobile security, in: SIGCSE '19, 2019.
[10] GData, https://www.gdatasoftware.com.
[11] Z. Lv, W. Mazurczyk, S. Wendzel, H. Song, Guest editorial: Recent advances in cyber-physical security in industrial environments, IEEE Trans. Ind. Inf. 15 (2019) 6468–6471.
[12] K. Xu, Y. Li, R.H. Deng, K. Chen, Deeprefiner: Multi-layer android malware detection system applying deep neural networks, in: 2018 IEEE European Symposium on Security and Privacy (EuroS&P), 2018, pp. 473–487.
[13] J. Ahn, Deep android malware detection, 2016.
[14] M. Amn, T. Ali, M. Tehseen, M.A. Khan, F.A. Khan, S. Anwar, Static malware detection and attribution in android byte-code through an end-to-end deep system, Future Gener. Comput. Syst. 102 (2020) 112–126.
[15] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, X. Zhang, Exploring permission-induced risk in android applications for malicious application detection, IEEE Trans. Inf. Forensics Secur. 9 (2014) 1869–1882.
[16] S. Chakradeo, B. Reaves, P. Traynor, W. Enck, Mast: triage for market-scale mobile malware analysis, in: WiSec '13, 2013.
[17] Y. Aafer, W. Du, H. Yin, Droidapiminer: Mining API-level features for robust malware detection in android, in: SecureComm, 2013.
[18] P. Faruki, V. Ganmoor, V. Laxmi, M.S. Gaur, A. Bharmal, Androsimilar: robust statistical feature signature for android malware detection, in: SIN, 2013.
[19] M.Y. Wong, D. Lie, Intellidroid: A targeted input generator for the dynamic analysis of android malware, in: NDSS, 2016.
[20] H. Cai, N. Meng, B.G. Ryder, D. Yao, Droidcat: Effective android malware detection and categorization via app-level profiling, IEEE Trans. Inf. Forensics Secur. 14 (2019) 1455–1470.
[21] S. Chen, M. Xue, Z. Tang, L. Xu, H. Zhu, Stormdroid: A streaminglized machine learning-based system for detecting android malware, in: ASIA CCS '16, 2016.
[22] M. Lindorfer, M. Neugschwandtner, C. Platzer, Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis, in: 2015 IEEE 39th Annual Computer Software and Applications Conference, Vol. 2, 2015, pp. 422–433.
[23] P. Vinod, A. Zemmari, M. Conti, A machine learning based approach to detect malicious android apps using discriminant system calls, Future Gener. Comput. Syst. 94 (2019) 333–350.
[24] I. Firdausi, C. Lim, A. Erwin, A.S. Nugroho, Analysis of machine learning techniques used in behavior-based malware detection, in: 2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies, 2010, pp. 201–203.
[25] D. Ucci, L. Aniello, R. Baldoni, Survey of machine learning techniques for malware analysis, Comput. Secur. 81 (2019) 123–147.
[26] P. Feng, J. Ma, C. Sun, X. Xu, Y. Ma, A novel dynamic android malware detection system with ensemble learning, IEEE Access 6 (2018) 30996–31011.

[27] H. Fereidooni, M. Conti, D. Yao, A. Sperduti, Anastasia: Android malware detection using static analysis of applications, in: 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), 2016, pp. 1–5.
[28] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, Drebin: Effective and explainable detection of android malware in your pocket, in: NDSS, 2014.
[29] M. Zhang, Y. Duan, H. Yin, Z. Zhao, Semantics-aware android malware classification using weighted contextual API dependency graphs, in: CCS '14, 2014.
[30] E. Mariconti, L. Onwuzurike, P. Andriotis, E.D. Cristofaro, G.J. Ross, G. Stringhini, Mamadroid: Detecting android malware by building Markov chains of behavioral models, in: NDSS 2017, 2016.
[31] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, K.-P. Wu, Droidmat: Android malware detection through manifest and API calls tracing, in: 2012 Seventh Asia Joint Conference on Information Security, 2012, pp. 62–69.
[32] L.-K. Yan, H. Yin, Droidscope: Seamlessly reconstructing the OS and dalvik semantic views for dynamic android malware analysis, in: USENIX Security Symposium, 2012.
[33] A. Saracino, D. Sgandurra, G. Dini, F. Martinelli, Madam: Effective and efficient behavior-based android malware detection and prevention, IEEE Trans. Dependable Secure Comput. 15 (2018) 83–97.
[34] M. Spreitzenbarth, T. Schreck, F. Echtler, D. Arp, J. Hoffmann, Mobile-sandbox: combining static and dynamic analysis with machine-learning techniques, Int. J. Inf. Secur. 14 (2014) 141–153.
[35] H. Cai, X. Fu, A. Hamou-Lhadj, A study of run-time behavioral evolution of benign versus malicious apps in android, 2020.
[36] Z. Yuan, Y. Lu, Z. Wang, Y. Xue, Droid-sec: deep learning in android malware detection, in: SIGCOMM 2015, 2015.
[37] X. Su, D. Zhang, W. Li, K. Zhao, A deep learning approach to android malware feature learning and detection, in: 2016 IEEE Trustcom/BigDataSE/ISPA, 2016, pp. 244–251.
[38] X. Su, W. Shi, X. Qu, Y. Zheng, X. Liu, Droiddeep: using deep belief network to characterize and detect android malware, 2020.
[39] T. Kim, B. Kang, M. Rho, S. Sezer, E.G. Im, A multimodal deep learning method for android malware detection using various features, IEEE Trans. Inf. Forensics Secur. 14 (2019) 773–788.
[40] Z. Yuan, Y. Lu, Y. Xue, Droiddetector: android malware characterization and detection using deep learning, 2016.
[41] S. Wang, Z. Chen, Q. Yan, K. Ji, L. Peng, B. Yang, M. Conti, Deep and broad URL feature mining for android malware detection, Inform. Sci. 513 (2020) 600–613.
[42] E.B. Karbab, M. Debbabi, Maldy: Portable, data-driven malware detection using natural language processing and machine learning techniques on behavioral analysis reports, Digit. Investig. 28 (2019) S77–S87.
[43] Y. Kim, Convolutional neural networks for sentence classification, in: EMNLP, 2014.
[44] Y. Zhou, X. Jiang, Dissecting android malware: Characterization and evolution, in: 2012 IEEE Symposium on Security and Privacy, 2012, pp. 95–109.
[45] A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, Bag of tricks for efficient text classification, 2017, ArXiv:abs/1607.01759.