Performance Comparison of Android Malware Detection Methods

Yuandi Wang1,*

1International School, Beijing University of Posts and Telecommunications, Beijing, 100000, China

*2018212655@bupt.edu.cn

Abstract. With the growth of Android apps market share, there are more and more kinds of malicious apps for Android system. At the same time, the method for malware detecting is evolving. In order to deal with the growing threat of malware, many detecting methods are combined with machine learning algorithms. In this paper, we compared the ability to detect malware of five algorithms on a dataset about Android apps networking. The result revealed that the networking data could be used as a reference for the classification of Android apps. Moreover, among the five algorithms used in this paper, the ensemble learning algorithm using the decision tree as base learner performed the best in the comparison. It reached more than 75% accuracy when predicting malware using the test set.

1. Introduction

Android has dominated the smartphone operating system (OS) markets, with more than 70% of the total share until November 2020 [1]. The total number of Android apps on Google Play has reached 3 million [2]. Moreover, there are still many apps on third-party Android markets. Different from other OS, like IOS, Android gives consumers an option to download apps from third-party markets. Google did not proactively detect or remove potential malicious apps until users report them as malware. This mechanism makes the Android ecosystem more flexible. However, it also introduces many risks. These features together attract many unfriendly software developers. Users may face more risks since they might accidentally download and give permissions to harmful apps. In such circumstances, users need additional reliable suggestions on whether the apps can be trusted or not.

There have been lots of methods applied for detecting Android malware [3-5]. There are two main steps in the detection of Android malware; one is feature extraction, and the other is classification. According to the processing data of applications in the "classification" step, malicious application detection technology can be divided into three types: static analysis, dynamic analysis and hybrid analysis [6]. Static analysis does not need the execution of the application, which the dynamic analysis requires the application to run in order to detect the maliciousness. And the hybrid analysis combines static and dynamic analysis.

Machine learning based detecting methods is usually static or hybrid. The machine learning process is static, but the data used in the learning process may need an extract from dynamic execution. For example, the malicious app may have different intents, permissions, component deployment, and APIs in their manifest files [7]. These data can be extracted from the manifest files. For other kinds of data, like networking data, will be generated after execution.

In this paper, the second kind of data, which is the networking data, to implement malware detection, is utilized. From a networking perspective, malicious apps and benign apps have different network
behaviors, and they may generate different data when communicating through the internet layers [8]. As Malwares usually have different intentions from benign software, they may also have different internet communication characteristics. For instance, malware may want to send some extra private data back to the server, and this irregular activity will leave a noticeable trace. Data from transport and network layer, such as packet length, flags, headers, flow speed etc., could be distinguished for malware detection. The information that determines whether they are malware or not is hidden under these networking data.

Five different machine learning algorithms are tested, which are Classification and Regression Tree (CART) [9], Random Forest (RF) [10], XGBoost [11], K-nearest Neighbor (KNN), and Deep Neural Network (DNN), based on the network data of Android apps. It would be difficult to use traditional methods to exploit the underlying features of this kind of data. Machine learning algorithms can be taken to train a relatively proper model to make good use of these data.

After training by the networking data, these five models gave at least more than 60 percent accuracy. Different algorithms have different adaptability to this data. Random forest is the most accurate model in processing this dataset. XGBoost is an ensemble model using CART decision tree; however, it does not improve the accuracy much. Deep neural network requires the most computing time while the result is not good. KNN also performs well.

The rest of this paper is organized as follows: section 2 reviews the related work in Android malware detecting; the methods used in this paper were introduced in Section 3; the experiment results are discussed and analyzed in Section 4; a conclusion is drawn in Section 5.

2. Related Work

Numerous approaches based on dynamic or static analysis to detect Android malicious software on a mobile device has been established. Xu et al. established a solution that can repackage arbitrary applications to attach user-level sandbox and policy enforcement code to watch any malicious behavior [12]. Zheng et al. proposed SmartDroid that extracted data by analyzing both Activity and Function Call Graphs, and then uses dynamic analysis to detect UI interactions related to malicious activities [13]. Dixon et al. focused on monitoring the abnormal usage of hardware, for example, power loss and memory consumption [14]. There is also an approach base on behavior analysis, compare application behaviors with test malware from a known dataset. Burguera et al. utilized a dynamic detector embedded in an overall framework to collect traces of real users based on crowdsourcing. And the framework was established by analyzing both artificial malware and real malware [15]. Another general approach is the static analysis approach. For example, Christodorescu et al. combined instruction semantics into the pattern-matching approach to detect malicious traits [16]. And Lo et al. proposed a Malicious Code Filter (MCF) to distinguish malicious code from benign code [17]. This method bases on the use of tell-tale signs, which are program properties about whether the codes are benign or not.

Some other researches applied more complex detection on malware. Dini et al. proposed a multi-level anomaly detector for Android malware. It concurrently monitors both kernel-level and user-level. This helps this detector to detect malware contained in unknown applications [18]. Saxe et al. proposed a deep neural network by binaries sourced directly from customers and internal malware databases [19]. Su et al. designed an Android malware detection approach called DroidDeep consisted of four components, including feature extraction, feature transformation, deep learning model and classification [20]. In the paper of Yuan et al., more than two hundred features from both static analysis and dynamic analysis are used in deep learning method [21]. Li et al. mined the permission data of Android apps to find the most important permissions for distinguishing malicious apps. They then use several machine-learning-based classification algorithms to classify different families of malware and benign apps. They found that machine learning algorithms with a tree structure, usually build better malware detection compared to others.

3. Method

We chose five typical machine learning algorithms for this malware detection, which is a kind of binary classification. These five algorithms were applied individually on the same networking dataset.
3.1 Classification and Regression Tree
A decision tree is a common algorithm for the classification task. The key to this algorithm is to select the best feature for each node. This feature will decide the purity of samples in the child node. CART uses Gini coefficient as the criterion to select features. The Gini coefficient represents the impurity of the model. The smaller the Gini coefficient is, the lower the impurity is, and the better the features are. This feature of CART helps a lot in the classification task. As there are more than eighty features in the dataset we use, it is quite important to focus on the more relevant features to the label.

3.2 Random Forest
Random Forest (RF) is a kind of ensemble learning algorithm. It is a model extended and modified from Bootstrap Aggregating (Bagging), representing a parallel ensemble learning method. This method bases on bootstrap sampling. Given a dataset with a total of m samples, randomly extract one sample into a sample set, and then put it back. Repeat this procedure m times to get a sample set with m samples. In this way, we can get T sample sets. Train single base learners on each sample set individually and combine all T base learners to get the final model. To combine all the prediction results of each base learner, Bagging usually uses a voting method in the classification task. For RF, the base learner is a decision tree. Compared with the traditional decision tree, the RF decision trees will randomly select a part of all features at each node and select the best feature from the selected feature set.

3.3 XGBoost
XGBoost stands for Extreme Gradient Boosting and it comes from Boosting algorithm, which is also an ensemble learning method. Its base learner is decision tree. All Boosting algorithms have similar mechanisms. It uses the given dataset to train a base learner and adjust the distribution of the dataset according to the performance of the base learner. Repeat this step T times can we get T base learners. The result is the weighted combination of these T base learners.

XGBoost is basically a Gradient Boosting Decision Tree (GBDT). The core of GBDT is that its objective function is minimized by gradient descent of function space. The improvements of XGBoost are mainly on the calculation of loss function. The traditional Gradient Boosting (GB) only uses the first derivative information in the optimization, while XGBoost uses the second-order Taylor expansion for the cost function. Both first and second derivatives are used at the same time. It adds a regular term $\Omega(f_k)$ to the objective function is an important improvement of XGBoost, so that each iteration is regularized at the mathematical level, and the effect and operability are greatly improved. The object function of XGBoost is as follow:

$$L(\phi) = \sum_{i} l(\hat{y}_i, y_i) + \sum_{k} \Omega(f_k)$$  \hspace{1cm} (1)

Object function of the t round:

$$L^{(t)} = \sum_{i=1}^{n} [l(y_i, \hat{y}_{t-1}^{(t)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$  \hspace{1cm} (2)

where

$$g_i = \hat{\partial}_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \quad \text{and} \quad h_i = \hat{\partial}^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$  \hspace{1cm} (3)

3.4 Deep Neural Network
Deep Neural Network (DNN) contains three parts, which are input layer, hidden layers and output layer. Each layer has multiple neurons, and these neurons can be seen as threshold logic units. As shown in Figure 1, it is an example of a simple neural network with four layers including input layer and output layer.
Neuron can calculate the input from other neurons of the former layer through weighted connection. And use an activation function to manage the result of the neuron. Taking Sigmoid function as an example, it can map a large range of output values into (0,1), as shown in Figure 2. This feature can be useful in the classification task. A single layer of the functional neuron can only deal with linearly separable problems. Adding more hidden layers will make the neural network solve more complex problems. To make the DNN structure learn the dataset, the Back Propagation (BP) algorithm is applied to it. BP based on gradient descent strategy, adjusting the connection weight between every neuron along the negative gradient direction of the target. In this paper, we added 25 hidden layers in the network and each layer has 200 nodes.

3.5 k-Nearest Neighbor
K-Nearest Neighbor (KNN) is a representation of lazy learning, which means it just save the training dataset without process of learning and wait until the test sample is received. Its working mechanism is relatively simple. For a given test sample, it will find k-nearest sample of the same category and the different category from the training dataset by some kinds of distance measure. For a classification task, the test result will decide by ‘voting’, namely, select the most frequent label among the k neighbors. The distance measure can be weighted to make the sample near the target point influence the result. KNN is good at predicting and automatically classification large samples, so it is a suitable algorithm for the malware detection test.
As shown in Figure 3, the black triangle represents a new sample waiting to be classified. And each green circle, blue rectangle and purple diamond represent one sample from each category in the training set. When \( k=13 \), the outer circle is the reference range, and there are most purple diamonds in the outer circle. Therefore, the new sample will be classified as a purple diamond. When \( k=3 \), the range narrows the scope to the inner circle, and the new sample belongs to the green circle this time.

### 4. Experiment results and analyze

#### 4.1 Dataset description

The dataset used in this paper is CICAndMal2017 [23], which extracts more than eighty features about network traffic. These features are captured during three stages, which are installation, before start and after start. This data set can comprehensively show the characteristics of different Android Software in network connection. In this research, we selected 68 features from a total of 81 features for training. For data of each feature, we applied zero-mean normalization to normalize it. The data sample is shown in Table 1.

| Source | Port | Bwd IAT Total | Destinati on Port | Protocol | … | Idle Max | Fwd Packets/s | Bwd IAT Std |
|--------|------|----------------|-------------------|----------|---|----------|---------------|-------------|
| Benign | 80   | 11065337       | 42047             | 6        | … | 0        | 0.088435      | 0           |
|        | 40573| 0              | 443               | 6        | … | 0        | 26.24603      | 0           |
|        | 57027| 431            | 443               | 6        | … | 0        | 26.74726      | 0           |
|        | 34996| 0              | 443               | 6        | … | 0        | 9.469024      | 0           |
|        | …    | …              | …                 | …        | … | …        | …             | …           |
| Malware| 53491| 53491          | 106982            |          |    |          |               |             |

#### 4.2 Analysis of experimental results

As shown in Table 2, compared with the other four algorithms, the performance of DNN is not satisfactory. It only achieved an accuracy of 68.06% on the test set. Meanwhile, DNN required the most computational power and time among these five algorithms. KNN gained a similar accuracy as DNN on the test set, which is 69.81%. As a relatively simple classification method, this accuracy is enough. It spent much less time to finish the classification than DNN did. The CART achieved an accuracy of 73.97% on the test set. As a classic binary classifier, its performance is better than DNN and KNN. For
XGBoost, an ensemble learning algorithm, it is interesting that it failed to achieve significantly better accuracy than CART on the test set. Its accuracy is 73.14%, which is almost the same as CART. For another ensemble learning algorithm, which also uses the decision tree as the base learner, the RF did have a better performance. The accuracy of RF is the highest among these five algorithms, which is 76.44%.

Table 2. Overall comparison

| Algorithm | CART Precision | CART Recall | CART F1 | RF Precision | RF Recall | RF F1 | Error | Accuracy |
|-----------|----------------|-------------|---------|--------------|-----------|-------|-------|----------|
| Benign    | 0.7353         | 0.7388      | 0.7370  | 0.7422       | 0.7440    | 0.7440| 0.2603| 0.7397   |
| Malware   | 0.6952         | 0.7046      | 0.6999  | 0.6963       | 0.7011    | 0.7011| 0.3019| 0.6981   |

5. Conclusion

In this paper, we try to detect Android malware using networking data and different machine learning algorithms. A total of five algorithms -- CART, RF, XGBoost, DNN and KNN were tested on a dataset about the networking of Android apps. The experiment results show that there is a certain relationship between network information and whether it is malicious software or not. Among these five algorithms, XGBoost and RF, two ensemble algorithms, and CART, a classical binary classifier, have achieved good results, and their accuracy in malware detection is over 70%. In contrast, KNN and DNN did not perform so well, and their accuracy is lower than 70%.

Reference

[1] Information from https://gs.statcounter.com/os-market-share/mobile/worldwide
[2] Information from: www.appbrain.com/stats/number-of-android-apps
[3] Felt, Adrienne Porter, et al. "Android permissions demystified." Proceedings of the 18th ACM conference on Computer and communications security. 2011.
[4] Maier, Dominik, Tilo Müller, and Mykola Protsenko. "Divide-and-conquer: Why android malware cannot be stopped." 2014 Ninth International Conference on Availability, Reliability and Security. IEEE, 2014.
[5] Li, Juanru, et al. "InDroid: An automated online analysis framework for Android applications." Crisis Intervention Team (CIT) (2014).
[6] Tam, Kimberly, et al. "The evolution of android malware and android analysis techniques." ACM Computing Surveys (CSUR) 49.4 (2017): 1-41.
[7] Wu, Dong-Jie, et al. "Droidmat: Android malware detection through manifest and api calls tracing." 2012 Seventh Asia Joint Conference on Information Security. IEEE, 2012.
[8] Morales, Jose Andre, et al. "Analyzing and exploiting network behaviors of malware." International conference on security and privacy in communication systems. Springer, Berlin, Heidelberg, 2010.
[9] Breiman, Leo, et al. Classification and regression trees. CRC press, 1984.
[10] Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.
[11] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. 2016.
[12] Xu, Rubin, Hassen Saïdi, and Ross Anderson. "Aurasium: Practical policy enforcement for android applications." Presented as part of the 21st {USENIX} Security Symposium (USENIX Security 12). 2012.
[13] Zheng, Cong, et al. "Smartdroid: an automatic system for revealing UI-based trigger conditions in Android applications." Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices. 2012.

[14] Dixon, Bryan, et al. "Location based power analysis to detect malicious code in smartphones." Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. 2011.

[15] Burguera, Iker, Urko Zurutuza, and Simin Nadji-Tehrani. "Crowdroid: behavior-based malware detection system for Android." Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. 2011.

[16] Cristodorescu, Mihai, et al. "Semantics-aware malware detection." 2005 IEEE Symposium on Security and Privacy (S&P'05). IEEE, 2005.

[17] Lo, Raymond W., Karl N. Levitt, and Ronald A. Olsson. "MCF: A malicious code filter." Computers & Security 14.6 (1995): 541-566.

[18] Dini, Gianluca, et al. "MADAM: a multi-level anomaly detector for Android malware." International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security. Springer, Berlin, Heidelberg, 2012.

[19] Saxe, Joshua, and Konstantin Berlin. "Deep neural network based malware detection using two dimensional binary program features." 2015 10th International Conference on Malicious and Unwanted Software (MALWARE). IEEE, 2015.

[20] Su, Xin, et al. "A deep learning approach to Android malware feature learning and detection." 2016 IEEE Trustcom/BigDataSE/ISP. IEEE, 2016.

[21] Yuan, Zhenlong, et al. "Droid-sec: deep learning in Android malware detection." Proceedings of the 2014 ACM conference on SIGCOMM. 2014.

[22] Li, Jin, et al. "Significant permission identification for machine-learning-based Android malware detection." IEEE Transactions on Industrial Informatics 14.7 (2018): 3216-3225.

[23] Lashkari, Arash Habibi, et al. "Toward developing a systematic approach to generate benchmark Android malware datasets and classification." 2018 International Carnahan Conference on Security Technology (ICCST). IEEE, 2018.