Information Sciences 000 (2017) 1–19

Contents lists available at ScienceDirect

Information Sciences

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

Machine learning based mobile malware detection using highly imbalanced network traffic

Zhenxiang Chen [a,b,*], Qiben Yan [c,**], Hongbo Han [a,b], Shanshan Wang [a,b], Lizhi Peng [a,b], Lin Wang [a,b], Bo Yang [b,**]

[a] School of Information Science and Engineering, University of Jinan, Jinan 250022, China
[b] Shandong Provincial Key Laboratory of Network Based Intelligent Computing, Jinan 250022, China
[c] University of Nebraska-Lincoln, Lincoln, NE 68588, USA

ARTICLE INFO

Article history:
Received 15 March 2016
Revised 8 April 2017
Accepted 28 April 2017
Available online xxx

Keywords:
Network traffic
Malicious apps
Imbalanced data
Malware detection
Machine learning

ABSTRACT

In recent years, the number and variety of malicious mobile apps have increased drastically, especially on Android platform, which brings insurmountable challenges for malicious app detection. Researchers endeavor to discover the traces of malicious apps using network traffic analysis. In this study, we combine network traffic analysis with machine learning methods to identify malicious network behavior, and eventually to detect malicious apps. However, most network traffic generated by malicious apps is benign, while only a small portion of traffic is malicious, leading to an imbalanced data problem when the traffic model skews towards modeling the benign traffic. To address this problem, we introduce imbalanced classification methods, including the synthetic minority oversampling technique (SMOTE) + support vector machine (SVM), SVM cost-sensitive (SVMCS), and C4.5 cost-sensitive (C4.5CS) methods. However, when the imbalance rate reaches a certain threshold, the performance of common imbalanced classification algorithms degrades significantly. To avoid performance degradation, we propose to use the imbalanced data gravitation-based classification (IDGC) algorithm to classify imbalanced data. Moreover, we develop a simplex imbalanced data gravitation classification (S-IDGC) model to further reduce the time costs of IDGC without sacrificing the classification performance. In addition, we propose a machine learning based comparative benchmark prototype system, which provides users with substantial autonomy, such as multiple choices of the desired classifiers or traffic features. Using this prototype system, users can compare the detection performance of different classification algorithms on the same data set, as well as the performance of a specific classification algorithm on multiple data sets.

1. Introduction

With the rapid growth of mobile networks, the number of mobile devices surges to a record high. However, mobile users still face serious threats from ransomware, spyware, malicious apps, and financial malware. Modern mobile devices are no longer confined to traditional communication services, and the popularity of apps related to e-commerce, per-

* Corresponding author at: School of Information Science and Engineering, University of Jinan, Jinan 250022, China.
** Corresponding authors.
E-mail addresses: czx@ujn.edu.cn (Z. Chen), yan@unl.edu (Q. Yan), yangbo@ujn.edu.cn (B. Yang).

http://dx.doi.org/10.1016/j.ins.2017.04.044
0020-0255/

sonal payments, and social communication on mobile devices is continuously increasing. Security issues in mobile devices, particularly app security on Android platforms, cannot be overlooked [19]. Several practical methods have been proposed, and the existing malware detection methods can be roughly classified into three categories: static analysis, dynamic system-level behavior analysis, and network-level behavior detection [2].

Static analysis and dynamic system-level behavior analysis are common methods used to detect malicious apps. Static analysis utilizes reverse-engineering techniques to analyze apps' codes, which relies on semantic signatures and focuses on analyzing code snippets without executing them. It extracts static features from malicious apps, including binary sequences, sequences of function calls, request permission sequences, and any other static information to determine whether an app exhibits malicious behavior. These methods are more resilient to changes in malware codes, but they can be easily disrupted by obfuscation techniques [14]. Dynamic system-level behavior analysis [13,42] focuses on extracting reliable information from the execution traces of malicious apps. The downside of the dynamic analysis is the necessity to run the codes in a restricted environment that may influence malware behavior, and the difficulty in tracing the behavior back to the exact code location.

Recently, some researchers have begun exploring how malicious apps can be detected using the network traffic [16]. Most apps, whether benign or malicious, require network connectivity. Therefore, critical traces of malicious apps behavior can be identified by analyzing network traffic. A number of traffic features have been identified for malicious apps detection [10,16,44]. However, the prior arts have encountered some challenges, such as the lack of sufficient labeled traffic data, and the imbalanced nature of network traffic data, i.e., the proportion of benign traffic data is significantly higher than that of malicious traffic data in real world. These challenges lead to another serious issue, i.e., various methods are not fairly compared since they are usually evaluated against different date sets and evaluation criteria.

To address these challenges, we first capture Android malware traffic during the first few minutes to be used as training and testing data, and then we conduct malicious behavior detection using several common machine learning models, such as SVM, C4.5 and Naivebayes. Experimental results show that a part of malicious behaviors can be detected based on network traffic using the selected features. However, they also show the deficiencies of common machine learning methods to cope with the imbalanced data. Furthermore,we implement several common imbalanced data classification methods for malicious behavior detection, such as the synthetic minority oversampling technique (SMOTE) [7] + support vector machine (SVM), SVM cost-sensitive (SVMCS) [39], and C4.5 cost-sensitive (C4.5CS) [38] on the same traffic dataset. Experimental results show that, although the imbalanced machine leaning methods can improve the detection accuracy, these algorithms fall short when the degree of imbalance reaches a certain threshold.

Then, the imbalanced data gravitation-based classification (IDGC) [33] algorithm is applied to the traffic dataset, and the results show that IDGC performs better than the proposed common imbalanced algorithms on highly imbalanced traffic data. However, the practicality of the IDGC method is limited by its high time complexity. In order to improve the IDGC algorithm, we propose the simplex IDGC (S-IDGC) algorithm, which preserves the advantages of the IDGC model while reducing the time complexity significantly. The experimental results show that S-IDGC is capable of real-time Android malware behavior detection. In addition, we propose a machine learning based comparative benchmark prototype system, which enables the comparison of multiple classifiers on the same data set.

The objective of this work is to reveal the highly imbalanced issue for modeling the benign and malicious traffic. Additionally, the paper proposes a novel imbalanced classification model S-IDGC,which achieves superior performance in malware detection using network traffic, effectively addressing the highly imbalanced classification problem. Furthermore, we design a comparative benchmark prototype system to have a fair comparison among different machine learning based methods for malware behavior detection, which can be deployed as a public service for fostering research in this area.

The rest of this paper is organized as follows. Section 2 presents related work. Section 3 analyzes mobile malware traffic collection and the imbalanced problem. Machine-learning based Android malware detection is discussed in Section 4. Section 5 introduces the simplex imbalanced data gravitation classification model. In Section 6, a machine learning-based comparative benchmark prototype system is presented. Section 7 concludes the paper.

2. Related work

Recently, researchers strive to effectively identify malicious network traffic and determine the malicious apps of the corresponding traffic. One study [22] provides an in-depth analysis of the Internet infrastructure used for mobile malware detection. Other studies [28,43,45] analyze the dependencies between network traffic and the activities of users, which is a common approach for anomaly detection. A network-level behavioral malware clustering system is presented in [34], which focuses on detecting HTTP-based malware based on the structural similarity in malicious HTTP traffic generated by malware samples in the same family. In addition, a number of studies have introduced machine learning methods to detect malicious apps [4,29,41]. Network traffic features are modeled to identify malicious apps. For example, a machine learning based flow-level traffic classification system is designed to solve the large-scale network traffic classification problem [21]. Tenenboim-Chekinaet, et al. [36] extract mobile network-level features based on the cluster analysis of the behavior disparities of different apps, to identify the network behavior of malicious apps. Five machine learning classifiers are evaluated to detect malicious activities [15] using the network traffic of a mobile device. The most prominent botnet detection methods are presented in [37], which utilize machine learning to identify the malicious network behavior.

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044

Table 1
Top 24 families in our dataset (number ≥ 25).

ID	FamilyName	Num	ID	FamilyName	Num
1	FakeInstaller	925	13	ExploitLinuxLotoor	70
2	DroidKungFu	667	14	Glodream	69
3	Plankton	625	15	MobileTx	69
4	Opfake	613	16	FakeRun	61
5	GinMaster	339	17	SendPay	59
6	BaseBridge	330	18	Gappusin	58
7	Iconosys	152	19	Imlog	44
8	Kmin	147	20	SMSreg	41
9	FakeDoc	132	21	Yzhc	37
10	Geinimi	92	22	Jifake	29
11	Adrd	91	23	Hamob	28
12	DroidDream	81	24	Boxer	27

However, the existing methods based on machine learning to identify malicious apps, do not consider imbalanced traffic data. Imbalance occurs when the proportion of instances in one class (majority class) is significantly larger than that of instances in another class (minority class). Imbalanced classification is an important machine learning research topic that troubles most general classification models because of the imbalanced class distribution. Recently, a considerable amount of studies have focused on the problem of imbalanced data, and numerous scholars and engineers have tried to improve the problems [12,17,30,35]. Although no existing work focuses on the imbalance issue of network traffic for malware detection, several recent studies focus on imbalanced traffic classification [25,32] and spam detection [23]. For malware detection, collecting a sufficient number of malware samples to train a detection model is difficult, whereas only a small portion of network traffic can be labeled as malicious, which is the focus of this research. Malicious network traffic exhibits a series of characteristics that the traditional classification pattern does not consider [20], such as the data scarcity, noisy data, and data imbalance. To address the imbalanced data issue, we need to develop an effective highly imbalanced classification model.

A series of techniques have been proposed to carry out the imbalanced classification [5]. López et al. [26,27] analyze the technical trends of various imbalanced classification studies. These techniques are classified into three types of data-level approaches, cost-sensitive learning and ensemble learning.The data-level approaches are commonly used to solve the problem of imbalanced distribution of data samples, which alter the distribution of samples by resampling methods, including oversampling and undersampling method. The most famous oversampling method is the synthetic minority oversampling technique (SMOTE) [7], which reduces the class imbalance by creating new minority class samples. In contrast to oversampling methods. Undersampling methods rebalance class distributions by reducing the number of majority class samples [24]. Another common approach is cost-sensitive learning, which utilizes a cost matrix to train a classifier. The consequence of misclassification of minor class samples is worse than that of major class samples. By allocating different penalty coefficients to various classes, this method accurately recognizes the minority class samples, that are highly valuable.Classical classification models, such as C4.5 decision trees and SVM have been modified to C4.5CS [38] and SVMCS [39] using cost-sensitive learning. Ensemble learning is another widely-used method that is similar to the data-level approach. It trains the classifier on a subset, constructs a set of classifiers, and classifies new data points using a (weighted) vote of predictions. For example, Chawla et al. [8] combined SMOTE with boost learning to develop the SMOTEBoost method. In addition, the IDGC [33] model also provides an effective method to solve imbalanced data problems. In this paper, we propose a simplex imbalanced data gravitation classification (S-IDGC) model, which effectively address the problem of several common imbalanced classification methods while significantly reducing the time complexity of imbalanced classification.

3. Mobile malware traffic collection and imbalanced issue analysis

3.1. Data set

1) *Data set of android malware samples*
We use 5560 malware samples from the Drebin Project [2] to generate the malicious traffic data set. These malware samples were collected over the period from August 2010 to October 2012. An overview of the top 24 malware families (with malware number exceeding 25) in the data set is showed in Table 1, including several families that are still actively distributed in the app markets.
2) *Data set of android Benign samples*
We designed a crawler to collect benign apps from Android app markets, such as Hiapk market, 91Play market, Baidu market, and Qihu360 market. The collected Android apps are then verified by more than 30 AV scanners, which is integrated in VirusTotal [40]. The benign apps are classified into 19 categories, such as antivirus, browser, communication apps, etc. For example, a small fraction of the apps from the Hiapk market, which have been collected in June 2015, are listed in Table 2.

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044

Table 2
An overview of Android benign samples from hiapk market.

ID	Category	Num	ID	Category	Num
1	AntiVirus	385	10	MediaAndVideo	290
2	Browser	244	11	Personalization	229
3	Communication	116	12	Photography	237
4	DailyLife	385	13	Productivity	581
5	Education	320	14	Reading	343
6	Finance	324	15	Personalization	279
7	Games	1328	16	Social	208
8	HealthAndFitness	328	17	Tools	250
9	Input	228	18	TravelAndLocal	103

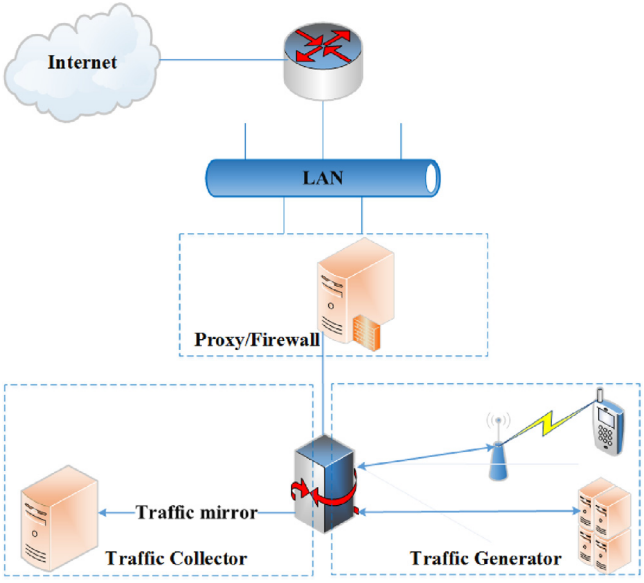


Fig. 1. Mobile traffic generation and capturing platform.

3.2. Automated traffic collection platform

To obtain malware traffic traces, we designed a practical traffic generation and monitoring platform [9]. As shown in Fig. 1, the platform consists of four parts: foundation platform, traffic generator, traffic collector and network proxy/firewall. The foundation platform provides an Android simulation environment and a command line mode of interaction, which offers basic functionalities such as simulator creation, app installation, and operation. The traffic generator is designed to install and activate malware samples to generate traffic traces automatically, which include two modules: automatic traffic generator and malware execution controller. The former module implements automatic installation and activation of a malware sample, whereas the latter is designed to automatically capture inbound and outbound traffic using tcpdump. We apply traffic mirroring technology to copy the traffic that passes through the gateway into a server. A network transport proxy/firewall is deployed between the Internet and malware running environment to monitor and control the attack behaviors.

3.3. Mobile traffic collection

We collect the network traffic of 5560 malware samples for the first 5 min using the traffic collection platform. The captured raw data includes numerous irrelevant traffic, such as SSDP, DHCP, ARP, NBNS, IGMP, and SMB packets, which are all filtered out to improve the traffic quality. The basic statistics of the collected traffic for the top 24 families are presented in Table 3. The same platform is used to collect the benign apps' network traffic. An overview of Android benign traffic data generated by apps from the Hiapk market is shown in Table 4.

3.4. Malicious traffic flow identification

Meanwhile, we design and implement a tool to convert the captured mobile traffic packets into traffic flows. The conversion of the data packets captured by tcpdump into flows is crucial for malicious flow identification. Research result [46] shows that over 80% of malware apps utilize the repackaging technique to embed malicious codes into benign

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044

Table 3
Basic traffic data information of the top 24 families.

FamilyName	Outbound data			Inbound data		
	Number	Bytes	Average	Number	Bytes	Average
FakeInstaller	2670	328,033	245.72	2936	2,310,115	1573.65
DroidKungFu	81,007	9,071,299	223.96	159,289	211,139,962	2651.03
Plankton	57,761	10,074,698	348.84	49,753	18,934,687	761.15
Opfake	2296	383,653	334.19	2216	1,151,089	1038.89
GinMaster	10,460	1,866,191	356.82	9517	5,069,755	1065.41
BaseBridge	6739	1,279,561	379.75	4810	2,118,627	880.93
Iconosys	689	107,116	310.93	469	142,831	609.09
Kmin	3957	574,358	290.3	3623	2,084,298	1150.59
FakeDoc	19,953	2,477,636	248.35	23,450	19,987,416	1704.68
Geinimi	1137	164,238	288.9	833	210,340	505.02
Adrd	2551	415,861	326.04	1714	423,377	494.02
DroidDream	609	87,265	286.58	627	429,145	1368.88
ExploitLinuxlotoor	930	115,500	248.39	1213	1,257,704	2073.71
Glodream	1257	182,634	290.59	1085	581,333	1071.58
MobileTx	4198	779,809	371.51	2795	1,686,455	1206.77
FakeRun	2503	453,336	362.23	1985	435,815	439.11
SendPay	864	127,448	295.02	551	125,652	492.38
Gappusin	3343	494,288	295.72	3267	2,409,820	1475.25
Imlog	528	92,449	350.19	464	221,303	953.89
SMSreg	2381	291,571	244.91	3495	3,457,447	1978.51
Yzhc	277	25,478	183.96	487	592,553	2433.48
Jifake	199	23,904	240.24	296	306,170	2068.72
Hamob	5811	823,083	283.28	8250	7,697,534	1866.07
Boxer	118	13,919	235.92	152	154,061	2027.12
Total	212,238	30,253,328		283,277	282,937,489	

Table 4
An overview of the benign traffic data from hiapk market's apps.

Category	Outbound data			Inbound data		
	Number	Bytes	Average	Number	Bytes	Average
AntiVirus	5357	1,767,586	329.95	6675	4,314,141	646.31
Browser	123	51,817	421.27	124	69,864	563.41
Communication	570	217,205	381.06	583	366,368	628.41
DailyLife	2868	1,405,723	490.14	2980	1,840,103	617.48
Education	3517	1,651,154	469.47	3222	1,914,591	594.22
Finance	1929	857,346	444.45	2005	1,352,451	674.53
Games	3200	2,150,731	672.10	3145	2,150,731	683.85
HealthAndFitness	1457	601,519	412.84	1484	923,474	622.28
Input	1974	788,152	399.26	1826	865,516	473.99
MediaAndVideo	6593	3,450,731	523.39	6602	3,584,366	542.92
NewsAndMagazines	3311	1,443,726	436.03	3353	2,237,195	667.22
Personalization	933	306,132	328.11	747	458,590	613.90
Photography	1399	742,071	530.42	1372	806,805	588.05
Productivity	1976	1,066,844	539.90	2030	1,351,631	665.82
Reading	6680	1,069,333	160.07	6716	3,262,301	485.75
Social	4505	1,954,169	433.77	4775	2,367,845	495.88
Tools	380	170,121	447.68	372	217,712	585.24
TravelAndLocal	2479	1,153,457	465.29	2225	1,089,088	489.47
Total	49,251	20,847,817		50,236	29,172,772	

apps and disguise themselves as benign apps. Thus, most malware apps appear as "benign" ones. Obviously, the collected traffic will be comprised of both the benign and malicious traffic data. The converted flows can be used to identify "pure" malicious traffic, which is guaranteed to be generated by malicious codes. The conversion function can be used to merge data packets into flows, and extract malicious flows based on the destination IPs or domain names. The traffic data conversion and malicious flow extraction are depicted in Algorithm 1.

3.5. Highly imbalanced problem analysis

This section discusses the highly imbalanced problem in malicious flow detection. Our survey on 500 mobile users shows that one equipment out of 5.6 equipments is infected by malware on average, and an average Android user installs 95 apps. The traffic data we collected from 4600 malware samples (top 24 malware families in Drebin [2] data set) includes 20,806 flows, in which 6170 flows are identified as malicious. The ratio of the amount of benign flows to malicious flows is 2.37 (i.e.,

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044

```
Data: Network traffic data packets captured by tcpdump
Result: Malicious traffic flows generated by the malware
 Have a preprocessing on the network traffic data packets;
while the network traffic data packets not null do
    read the traffic data packets into memory;
    converge the data packets into n flows;
    if the network traffic flow exists then
        extract the destination IP address;
        utilize the third-part scanning services to detect the IP address;
        if the IP address is benign then
            extract the domain name from the host field in HTTP packets;
            if the domain name is benign then
                discard this flow;
            else
                retain this flow as malicious;
            end
        else
            retain this flow as malicious;
        end
    else
        write the error information into the log file;
    end
end
```

Algorithm 1: The extraction of malicious traffic flow.

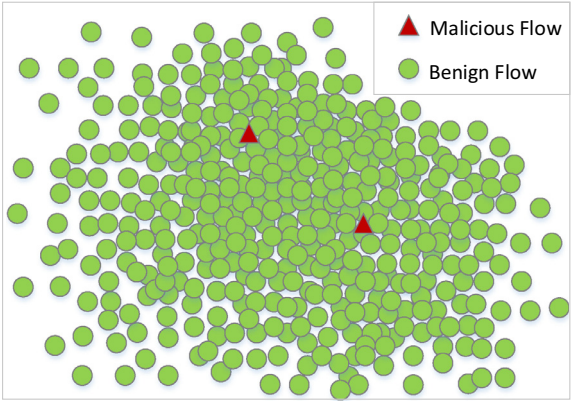


Fig. 2. The schematic diagram of quantity differences between benign flows and malicious flows .

(total flow number–malicious flow number)/malicious flow number). Based on large scale data analysis, we find that each malware samples generates 4 to 5 flows on average during the first 5 min, in which 1 to 2 of these flows are malicious. In comparison, each benign app generates 8.02 flows. Considering a real-world scenario, where one Android device is infected by one malware sample, whereas 5 other Android devices are safe. For each device, 95 benign apps are installed. Thus, these 6 Android devices will generate a total of 4565 (i.e., average flow generated by benign app $\times$ 95 $\times$ 6) flows. On the other hand, one malware sample in the infected device will generate one malicious flows.

The ratio of the amount of benign flows to malicious flows is approximately 4565: 1, which represents the imbalance ratio (IR) of the majority and minority class samples, as written in Eq. (1):

$$IR = \frac{n_{maj}}{n_{min}}, \tag{1}$$

where n_{maj} represents the flow number of the majority class samples (i.e.,benign traffic flow), and $n_{\min}$ denotes the flow number of the minority class samples (i.e.,malicious traffic flow). The average IR value reaches 4565, which indicates that only one in 4565 flows in a real network environment is malicious. To account for other factors affecting IR value, we consider the maximal IR value to be 7000 in the following experiments. The schematic diagram of the quantitative differences between the amount of benign flows and malicious flows is shown in Fig. 2, where we notice that there is a highly imbalanced problem for network flow based mobile malware detection.

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044

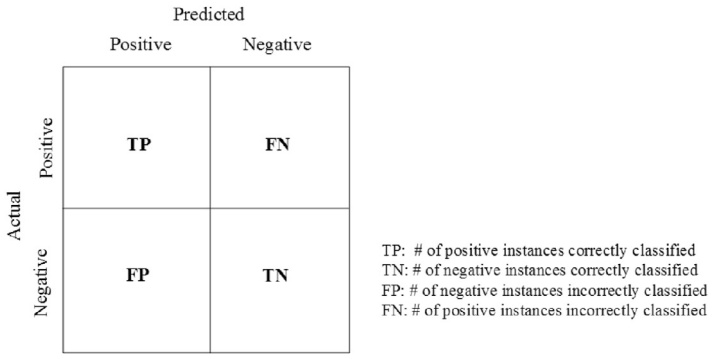


Fig. 3. Confusion matrix.

4. Machine learning-based android malware detection

4.1. Evaluation criteria

The detection accuracy rate is generally used to evaluate the performance of machine learning classifiers. The confusion matrix of a binary classification is presented in Fig. 3. True positive (TP) denotes the number of positive instances that are correctly classified, whereas false positive (FP) indicates the number of negative instances that are incorrectly classified as positive samples. True negative (TN) denotes the number of negative instances that are correctly classified, whereas false negative (FN) indicates the number of positive instances that are incorrectly classified as negative samples.

4.1.1. General evaluation criteria

In addition to the confusion matrix, the accuracy rate is typically used for evaluating the performance of classifiers.

- **Accuracy rate:** Based on the confusion matrix, the accuracy rate(ACC) is defined as:

$$ACC = \frac{TP}{TP + FP}. \tag{2}$$

4.1.2. Imbalanced evaluation criteria

The above-mentioned evaluation criteria all lean towards the majority class, while the result of minority class will be ignored. However, the minority class samples are essential for the malware detection problem that we are dealing with. Although the amount of malicious flows is few, but they are crucial for malware classification. ACC is not suitable for measuring the performance of imbalanced classification. Two commonly used imbalanced classification performance evaluation metrics are thus introduced below.

- **Area under curve**: The receiver operating characteristic (ROC) curve [6] is a 2D graphical illustration of the trade-off between the true positive rate (TPR) and the false positive rate (FPR), as shown in Fig. 4. TPR is also called sensitivity (Sens), whereas FPR is related to another general measure, namely, specificity (Spec), which is defined as follows:

$$Sens = \frac{TP}{TP + FN}, \tag{3}$$

$$Spec = \frac{TN}{TN + FP} = 1 - FPR. \tag{4}$$

The ROC curve shows the behavior of a classifier without considering class distribution or misclassification cost. The diagonal line represents the trade-off between sensitivity and one-specificity for a random model. For a classifier with excellent performance, the ROC curve should be at the farthest top left-hand corner. The Area Under Curve (AUC) is a commonly used evaluation measure for imbalanced classification. AUC is computed using the confusion matrix values: TPR and FPR.

$$AUC = \frac{1 + TPR - FRP}{2}. \tag{5}$$

Since $Spec = 1 - FPR$ and $Sens = TPR$, we get:

$$AUC = \frac{Sens + Spec}{2}. \tag{6}$$

The equation indicates that the AUC is the mean value of Sens and Spec, which balances the proportion of the correctly predicted positive and negative instances.

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044

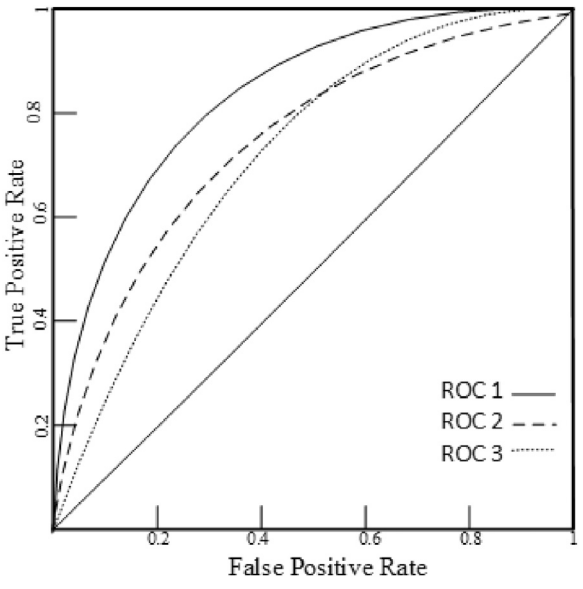
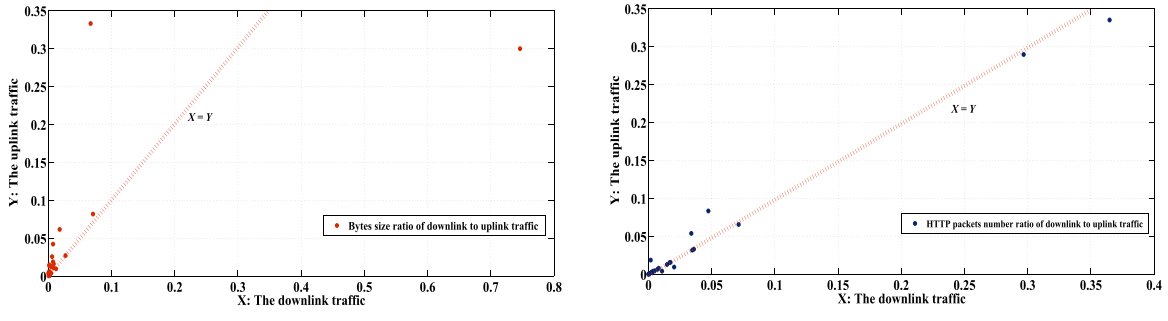


Fig. 4. Example of ROC curve.



(a) Packet number ratio of downlink to uplink traffic (b) Byte size ratio of downlink to uplink traffic

Fig. 5. The ratio of downlink to uplink traffic amount of the top 24 families.

- **Geometric mean**: GM is another method that has been used to compare positive and negative classes regardless of the class size [3]:

$$GM = \sqrt{Sens \times Spec}. \tag{7}$$

Similar to AUC, GM attempts to maximize the accuracy of each class in a balanced manner. GM is a performance metric that links both objectives. This method has been increasingly adopted in imbalanced classifications methods.

4.2. Extraction and analysis of flow features

In [9], we show that the amount of downlink traffic is different from that of uplink traffic as depicted in Fig. 5, where the x-axis represents downlink traffic, whereas the y-axis denotes uplink traffic. The red line represents $x = y$, which indicates that the amount of downlink traffic is equivalent to that of uplink traffic. Fig. 5(a) illustrates the packet number ratio of the downlink traffic to the uplink traffic. The packet amount ratio in most malware families falls above the red line. Fig. 5(b) describes the byte ratio of all the received data to all the sent data. The result indicates that the amount of uplink traffic is greater than that of downlink traffic, and most malware apps upload data more frequently than benign apps.

The current study focuses on extracting six statistic features from the amount of downlink traffic and the amount of uplink traffic, which are listed in Table 5. We will consider additional features in our future work.

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044

Table 5
The features extracted from flows.

Feature	Description
UpBytes	The number of bytes in uplink traffic flow
DownBytes	The number of bytes in downlink traffic flow
UpPkgNum	The number of upstream traffic flow packets
DownPkgNum	The number of downstream traffic flow packets
EveryUpBytes	The average number of bytes in uplink traffic flow packets
EveryDownBytes	The average number of bytes in uplink traffic flow packets

Table 6
Classic classifiers with their parameters.

Algorithm	Parameter
SVM	Kernel = PolyKernel, c = 1.0, epsilon = 1.0E-12, toleranceParameter = 0.001
C4.5	ConfidenceFactor = 0.25, reducedErrorPruning = False subtreeRaising = True, unpruned = False
Grading	MetaClassifier = ZeroR, numFolds = 10
Naivebayes	DisplayModelInOldFormat = False, useKernelEstimator = False useSupervisedDiscretization = False
BayesNet	Estimator = SimpleEstimator(-A 0.5), searchAlgorithm = K2(-P 1 -S BAYES)
Adaboost	NumIterations = 100, weightThreshold = 100

Table 7
Accuracy rate of the classical classifiers.

IR	BayesNet	SVM	C4.5	Grading	Adaboost	Naivebayes
100	0.995527	0.995455	0.999567	0.990116	0.999062	0.818123
200	0.997535	0.997009	0.999637	0.99507	0.998332	0.422346
300	0.998257	0.997458	0.996877	0.996732	0.998911	0.521017
400	0.998256	0.997747	0.999637	0.997529	0.999782	0.589984
500	0.998982	0.998182	0.999855	0.998037	0.998909	0.856844
600	0.998982	0.998545	0.999564	0.9984	0.998691	0.956424
700	0.998617	0.999272	0.999782	0.998617	0.999418	0.849232
800	0.999272	0.999272	0.999782	0.998763	0.999491	0.648133
900	0.998835	0.998999	0.999782	0.998908	0.999419	0.518597
1000	0.999418	0.999199	0.999709	0.999054	0.99949	0.935066

4.3. Evaluation on classic classifiers

4.3.1. Experiments

Six different classical machine learning classification algorithms are adopted to evaluate the different IR values of our collected mobile traffic data. The six classical classifiers are SVM, C4.5, Grading, Naivebayes, Bayesnet, and Adaboost, which are all implemented in Weka [18]. These classifiers are widely used machine learning algorithms, and the parameters of the six classical classifiers are listed in Table 6.

Considering the imbalanced problem for the IR values in the anomaly detection problem, the mobile traffic flow feature data set is divided into 10 groups. The corresponding IR values are 100, 200, 300, 400, 500, 600, 700, 800, 900, and 1000. These values represent the proportion of positive and negative samples in different network environments. A high IR value indicates that the amount of the positive samples corresponding to the majority samples is large. The distribution of the positive and negative samples is imbalanced.

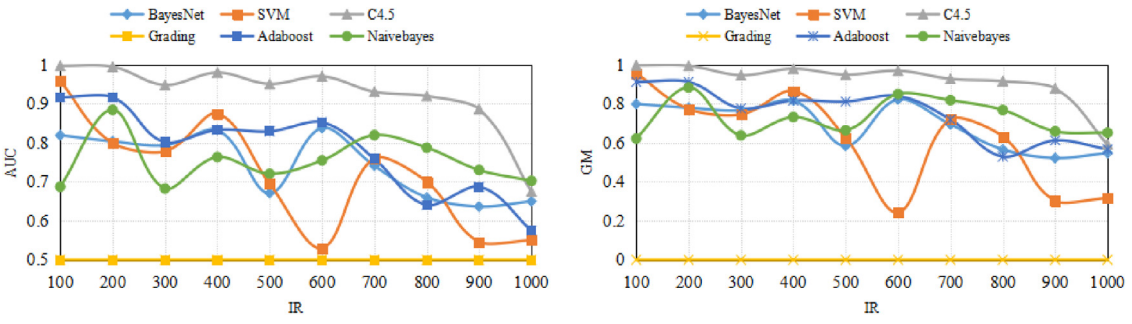
Each flow data set runs six classical classifiers with different IR values. The fivefold cross-validation method is used 10 times for the experiment, and the evaluation outcomes of ACC, AUC, and GM are obtained. The results are presented in Table 7 and Fig. 6(a) and (b).

4.3.2. Result analysis

Most of the classical classifiers in the ACC results exhibit excellent performance. The accuracy rate of these classifiers can reach over 99.9%, except for the Naivebayes classifiers. However, the AUC results show that the performance of all the classifiers decreases as the IR value increases. The AUC value of the five classical classifiers is below 0.8 when the IR value $\geq$ 600, except for the C4.5 classifier. However, the classifiers are not always improving when the IR value increases continuously. When the IR value $\geq$ 800, the AUC evaluation of C4.5 declines to 0.8 even when the classifiers exhibit the best performance. The AUC performance of all the classifiers tends to decrease, and most classifiers cannot reach 0.8 when IR $\geq$ 700.

The GM results indicate that IR = 600 is the threshold for most classifiers. The C4.5 works slightly better than the other classifiers. However, its performance declines when IR $\geq$ 800. The GM evaluation of all the classifiers decreases.

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044



(a) Comparative AUC results on multiple classifiers (b) Comparative GM results on multiple classifiers

Fig. 6. Evaluation results of the classical classifiers.

Table 8
Imbalanced classifiers with their parameters.

Algorithm	Parameter
SVMCS	KernelType = POLY, C = 100, eps = 0.001, degree = 1 gamma = 0.001, coef0 = 0.0, nu = 0.1, p = 1.0, shrinking = 1
C4.5CS	Pruned = TRUE, confidence = 0.25, instancesPerLeaf = 2, minimumExpectedCost = TRUE
SMOTE	Number of neighbors = 5, Type of SMOTE = both, Balancing = YES, ALpha = 0.5, Mu = 0.5
SVM	KernelType = POLY, C = 100, eps = 0.001, degree = 1, gamma = 0.001 coef0 = 0.0, nu = 0.1, p = 1.0, shrinking = 1
C4.5	Pruned = TRUE, confidence = 0.25, instancesPerLeaf = 2, minimumExpectedCost = TRUE
BalanceCascade	Pruned = True, confidence = 0.25, instancesPerLeaf = 2 Number of Classifiers in each Bag = 10, Algorithm = BALANCECASCADE
EasyEnsemble	Pruned = True, confidence = 0.25, instancesPerLeaf = 2 Number of Classifiers in each Bag = 10, Algorithm = BALANCECASCADE

The preceding analysis shows that the common classifiers work effectively on the extracted flow features. This result illustrates that malicious mobile traffic differs from benign mobile traffic. The extracted flow features are effective, and classical machine learning classifiers can be used to solve the malware detection problem. However, the performance of classical classifiers declines when the imbalanced problem gets worse.

4.4. Evaluation on imbalanced classifiers

4.4.1. Experiments

The preceding analysis shows that classical classifiers cannot perform efficiently due to the highly imbalanced problem. The performance of imbalanced classification methods is evaluated in this section. The selected algorithms are from KEEL [1], including SVMCS, C4.5CS, SMOTE + SVM, SMOTE + C4.5, BalanceCascade, and EasyEnsemble. The six imbalanced classifiers include the three most commonly-used methods for solving the imbalanced problem. SVMCS and C4.5CS are the commonly-used algorithms belonging to cost-sensitive learning. SMOTE + SVM and SMOTE + C4.5 are both commonly-used algorithms belonging to data-level learning. BalanceCascade and EasyEnsemble are both commonly-used algorithms belonging to ensemble learning. The parameters of the six classifiers are listed in Table 8.

To explore the feasibility of these classification methods on our mobile traffic flow data set, 16 sets of different network traffic environments, with IR values ranging from 100 to 7000, are simulated. Specifically, the IR values are set to 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 3000, 4000, 5000, 6000 and 7000. From the 16 sets of scenarios, data sets whose IR value ranges from 100 to 1000 are called low imbalanced data sets, whereas those whose IR value is greater than 1000 are called high imbalanced data sets. The initial minimum IR value in this research is 100, which represents an extremely imbalanced data set.

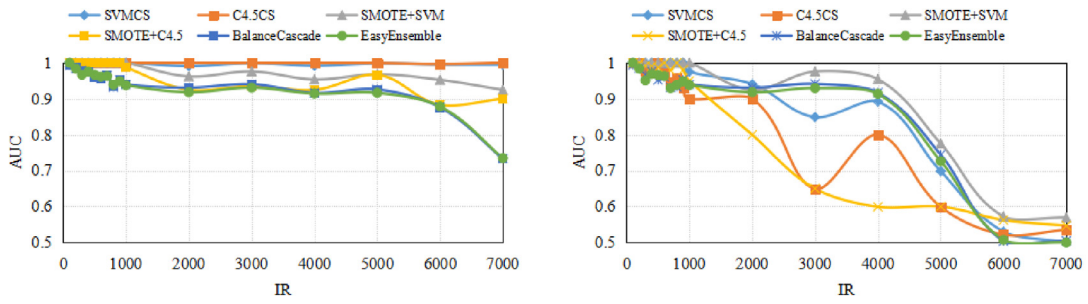
Based on the IR value, each group runs the six imbalanced classifiers on KEEL [1] with five-time and fivefold cross-validation. After training and testing, the AUC and GM evaluations of the six classifiers are derived. Fig. 7(a) and (b) show the AUC results on the training and testing sets. Fig. 8(a) and (b) present the GM results on the training and testing sets.

4.4.2. Result analysis

The AUC evaluations shown in Fig. 7 indicate that the classifiers based on the selected algorithms perform well. The AUC evaluation result does not decrease although the IR value reaches 7000 in the training data set. However, the testing results indicate that the AUC evaluations decline sharply when IR $\geq$ 4000. The performances of the C4.5CS algorithm and the SMOTE + C4.5 algorithm are worse than those of the other algorithms. The AUC testing evaluations of the algorithms also decrease when IR $\geq$ 2000. Although these algorithms exhibit ideal AUC results on the training data set, most of them still present reasonable AUC evaluation result when IR $\geq$ 4000.

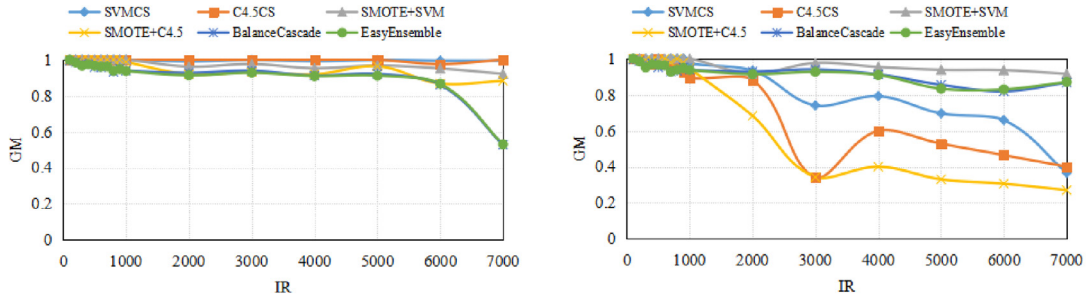
The GM evaluation results of the training data set in Fig. 8 show that most of the algorithms perform well, except for SMOTEBagging and SMOTEBoost. However, not all the GM evaluation results of the testing data set exhibit good

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044



(a) AUC evaluation results of the imbalanced classifiers on training set (b) AUC evaluation results of the imbalanced classifiers on testing set

Fig. 7. AUC evaluation results of the imbalanced classifiers.



(a) GM evaluation results of the imbalanced classifiers on training set (b) GM evaluation results of the imbalanced classifiers on testing set

Fig. 8. GM evaluation results of the imbalanced classifiers.

performance. Fig. 8(b) shows that SMOTE + SVM, BalanceCascade, and EasyEnsemble achieve better classification performance than SVMCS, C4.5CS, and SMOTE + C4.5.

The evaluations in terms of AUC and GM indicate that some of the selected imbalanced algorithms perform well in terms of GM. However, none of these algorithms performs efficiently when the IR value keeps increasing (all the AUC evaluation steps decline when IR $\geq$ 4000). These results indicate that within a certain imbalanced range, these methods can still achieve reasonable results, however, the performance of all the algorithms sharply declines as the IR value continuously increases.

5. Simplex imbalanced data gravitation classification model

5.1. The literature of IDGC model

Data gravitation-based classification (DGC) [31] is a nature inspired classification algorithm. Inspired by Newton's law of universal gravitation, DGC classifies data using gravitational comparison: for a testing instance, the gravitation from all classes of the training set are computed, then the testing instance is classified into the class which generates the largest gravitation. DGC has a set of theories simulating Newton law of universal gravitation that includes a number of concepts and lemmas. These concepts and lemmas support the classification principle of the DGC model.

The performance of DGC model is better than those of some other classical classification algorithms. However, the DGC model is less effective on an imbalanced data set because DGC classification has no specific mechanism to process minority class samples. Although the DGC classifier can achieve high accuracy, most of the minority class samples have been misclassified to the majority class. However, in some scenarios such as malicious flow detection, the contribution of the minority class samples is significantly higher than that of the majority class samples. Peng et al. [33] design an imbalanced data gravitation-based classification(IDGC) model, which uses the amplified gravitation coefficient (AGC) to modify the gravitation computation method, and rebalances the gravitational field strength disparity between the majority and minority classes. The IDGC model primarily aims at strengthening the gravitational fields of the minority class and weakening those of the majority classes.

The IDGC [33] model is analyzed and verified comprehensively by evaluating 44 data sets and comparing with other well-known imbalanced approaches, including SMOTE, SMOTEBagging, SMOTEBoost, EasyEnsemble, and SVMCS. The IDGC model performs fairly well. It provides an effective means to solving the imbalanced problems. However, the IDGC model

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044

uses particle swarm optimization (PSO) to optimize its feature weight coefficients. Thus, the computational complexity of IDGC is quite high. Although PSO can guarantee a global optimum, computational efficiency is a serious problem, especially in the real-time malicious flow detection with millions of users and massive network traffic data. Balancing quickly and achieving high precision at the same time is very difficult. On the other hand, the classifier model must be established quickly to achieve real-time detection. Thus, we need a method that can replace PSO which should satisfy the requirement of classification precision, weight coefficient computing time and real-time detection efficiency. We propose Simplex [11] to replace PSO while satisfying all the requirements.

5.2. S-IDGC model

5.2.1. Imbalanced gravitation for DGC model

An imbalanced data set is assumed to have k classes $c_1, c_2, \ldots, c_k$. For each class, $N_1, N_2, \ldots, N_k$ are the numbers of instances. The IDGC model introduces a new coefficient for class c_i, called the amplified gravitation coefficient (AGC) γ_i, $i = 1, 2, \ldots, k$, which is defined as follows:

$$\gamma_i = \frac{\sum_{j=1}^{k} N_j}{N_i}, \tag{8}$$

γ_i is the reciprocal of the proportion of the number of instances in class ci. Suppose that n features (inputs) exist in the target problem, and class c_i has l_i data particles with the following centroids: x_1, x_2, ... , and x_{l_i}. The testing instance (or position in problem space) $\mathbf{x}$=$[x_1, x_2, \ldots, x_n]$ is used γ to adjust the gravitation is written as follows:

$$F_i = [G_1, G_2, \ldots, G_{l_i}] \cdot \begin{bmatrix} \gamma_i \\ \gamma_i \\ \cdots \\ \gamma_i \end{bmatrix}, \tag{9}$$

where $G_j = \frac{m_j}{|r_j|^2}$ is the original gravitation from data particle j, and m_j is the data mass of the particle, and r_j is the Euclidean distance between the particle and testing instance/position. The weighted Euclidean distance r' is defined as follows:

$$r' = \sqrt{\sum_{i=1}^{n} w_i (x_{1i} - x_{2i})^2}. \tag{10}$$

Notably, $\mathbf{w} = [w_1, w_2, \ldots, w_n]^T$ is the feature weight vector in the n dimensional space. Therefore, Eq. (9) can be written as:

$$F_i = [\frac{m_1}{\sum_{j=1}^{n} w_j (x_{1j} - x_j)^2}, \frac{m_2}{\sum_{j=2}^{n} w_j (x_{2j} - x_j)^2}, \ldots, \frac{m_{l_i}}{\sum_{j=i}^{n} w_j (x_{ij} - x_j)^2}] \cdot \begin{bmatrix} \gamma_i \\ \gamma_i \\ \cdots \\ \gamma_i \end{bmatrix} \tag{11}$$

and

$$F_i = \sum_{k=1}^{l_i} \frac{\gamma_i m_k}{\sum_{j=1}^{n} w_j (x_{kj} - x_j)^2}. \tag{12}$$

The preceding formula is also used to calculate the gravitation of class c_i. Finally, the maximum value principle is applied to obtain the classification result of $\mathbf{x}$:

$$cls(\mathbf{x}) = ID(MAXF_1, F_2, \ldots, F_k), \tag{13}$$

where $ID(\cdot)$ is the operator that is used to obtain the class label value. The key of the model is to find a group of effective feature weights.

From the previous analysis, we can see that F_i is crucial in the IDGC model. As indicated in Eq. (12), F_i is dependent on $\mathbf{w} = [w_1, w_2, \ldots, w_n]^T$, which is optimized in the evaluation function. Given n dimensional feature weight coefficients $\mathbf{w} = [w_1, w_2, \ldots, w_n]^T$, the evaluation function is defined as:

$$fitness(\mathbf{w}) = \sum_{i=1}^{k} \gamma_i \frac{err_i}{N_i}. \tag{14}$$

5.2.2. Simplex based IDGC model

The simplex method [11] seeks the extremum of a multidimensional function and it does not use any derivative computation. The simplex method has the advantage of simple calculation and can be applied to a variety of unconstrained extreme problems. We use this simplex algorithm instead of the PSO in the IDGC model [33]. The $\mathbf{w} =< w_1, w_2, \ldots, w_n >$ represents the weight for each dimensional feature. The simplex algorithm primarily aims to obtain the minimum of the fitness function and determine the optimal weight coefficient $\mathbf{w}$. The details of S-IDGC model will be described as follows.

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044

1) **Initial simplex**: Initial simplex **S** is constructed by generating n vertices $w_1, \ldots, w_n$ around a given input point $w_{in} \in \mathbb{R}^n$, in which n indicates the n weight features. Here we set $w_1 = w_{in}$ to allow algorithm to initiate properly. The remaining $n-1$ vertices are then generated to obtain the standard shape of **S**, which is right-angled at w_1, and w_j is calculated as:

$$w_j := w_1 + h_j e_j, j = 1, \ldots, n, \tag{15}$$

where h_j is a stepsize in the direction of unit vector e_j in $\mathbb{R}^n$.

2) **Data gravitation**: After initiating the simplex, given the initial feature weight $\mathbf{w} = [w_1, w_2, \ldots, w_n]^T$, the data gravitation F_i is obtained according to Eq. (12).

3) **Classification error**: In the training set, the err_i is denoted as:

$$err_i = |lab(\mathbf{x_i}) - cls(\mathbf{x_i})|, \tag{16}$$

where the $lab(\mathbf{x_i})$ represents the actual class of the sample i in the training set. As we know, $cls(\mathbf{x_i})$ depends on the maximum gravity value of all classes $F_1, F_2, \ldots, F_i$ according to Eq. (13).

4) **Fitness function**: Considering Eqs. (8) and (14), the fitness evaluation function can be calculated as follows:

$$fitness(\mathbf{w}) = \sum_{i=1}^{k} N_i \sum_{i=1}^{k} \frac{err_i}{{N_i}^2}. \tag{17}$$

5) **Update w**: In the IDGC model, according to the fitness evaluation in Eq. (17), $fitness(\mathbf{w}) = f(\mathbf{w})$. After one iteration of simplex method to update **w**, optimal **w** is obtained, as described in Algorithm 2 .

Data: The preprocessed traffic flow data
Result: The optimal feature weight coefficients $\mathbf{w} =< w_1, w_2 \ldots, w_n >$
Initial the feature weight coefficient **w** in the simplex **S**;
while *The stoping criterion is not met* **do**
  Order the objective values at the vertices;
  Calculate the centroid w_{cen} of the vertices excluding w_{n+1};
  Reflection: Generate the reflection point $w_{ref} = 2w_{cen} - w_{n+1}$;
  if $f(w_{ref}) < f(w_1)$ **then**
    calculate the expansion point, $w_{\exp} = 2w_{ref} - w_{cen}$;
    if $f(w_{\exp}) < f(w_{ref})$ **then**
      set $w_{new} = w_{\exp}$;
    else
      set $w_{new} = w_{ref}$;
    end
  else if $f(w_1) \leqslant f(w_{ref}) < f(w_n)$ **then**
    set $w_{new} = w_{ref}$;
  else if $f(w_{n+1}) < f(w_{ref})$ **then**
    set $w_{new} = 2^{-1}(w_{n+1} + w_{cen})$;
  else
    set $w_{new} = 2^{-1}(w_{ref} + w_{cen})$;
  end
  From a new simplex by replacing the vertex w_{n+1} with w_{new};
  if $f(w_{new}) \geqslant f(w_{n+1})$ **then**
    form a new simplex by replacing all vertices $w^i, i = 0, 1, \ldots, n$, by $\bar{w}^i = 2^{-1}(w^i) + x_1$;
  end
end

Algorithm 2: One iteration of the simplex method.

6) **Optimal w**: If the fitness evaluation function reaches the stopping condition after several iterations, optimized **w** in the model is obtained and the class of the testing samples is calculated using Eq. (12).

7) **Testing result**: The optimized classifier is derived, and testing samples are utilized for evaluation according to Eq. (13) to get the classification results.

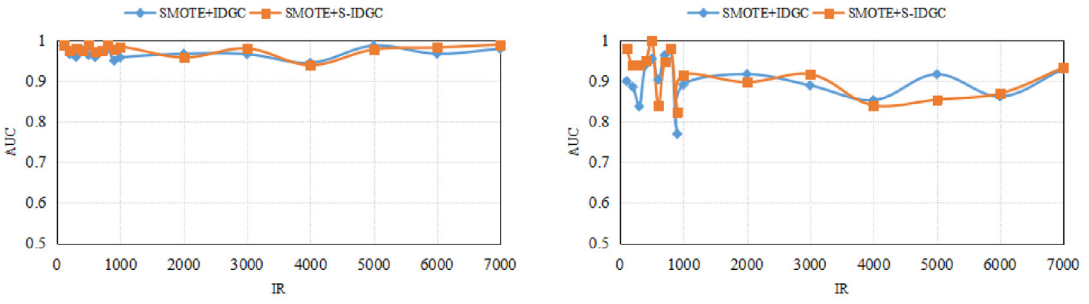
Evidently, according to Eq. (16), err_i is related to the independent variable **w**, which makes Eq. (16) a discontinuous and non-differentiable function. Thus, the function of the derivative cannot be used to get the optimal value. The fitness function requires at least n function evaluations per iteration.

For the nonlinear multi-dimensional optimization problem, S-IDGC uses the Nelder-Mead method to optimize the feature weight. The simplex method is adopted instead of PSO to optimize the feature weight coefficients, which obtains the minimum value by performing four types of operation, namely, reflection, expansion, contraction and shrinkage. The simplex method generally provides significant improvements for the first few iterations and rapidly produces satisfying results.

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044

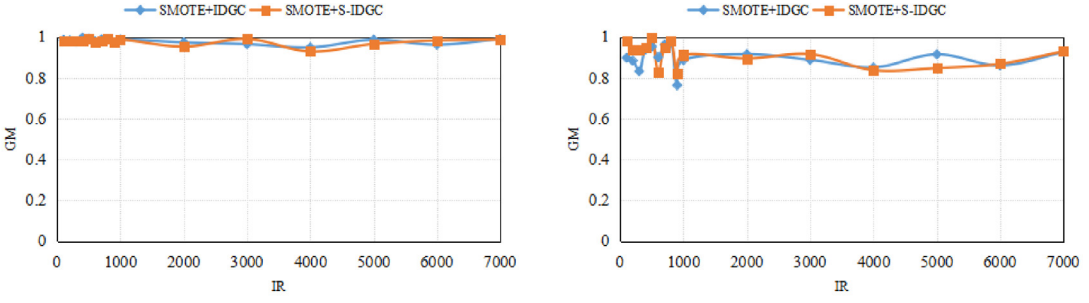
Table 9
Parameters of SMOTE+IDGC and SMOTE+S-IDGC classifiers.

Algorithm	Parameter
SMOTE	number of neighbors = 5, Type of SMOTE = both Balancing = YES, ALpha = 0.5, Mu = 0.5
IDGC	Radius of data particle = 0.01, PSO pop size = 30, MAX iteration time = 200
S-IDGC	Radius of data particle = 0.01, Simplex ccoeff = 0.5, ecoeff = 2.0 eps = 0.01, kconvage = 5, MAX iteration time = 200



(a) AUC evaluation of SMOTE+IDGC and SMOTE+S-IDGC on training set (b) AUC evaluation of SMOTE+IDGC and SMOTE+S-IDGC on testing set

Fig. 9. AUC evaluation of SMOTE+IDGC and SMOTE+S-IDGC.



(a) GM evaluation of SMOTE+IDGC and SMOTE+S-IDGC on training set (b) GM evaluation of SMOTE+IDGC and SMOTE+S-IDGC on testing set

Fig. 10. GM evaluation of SMOTE+IDGC and SMOTE+S-IDGC.

This S-IDGC model exhibits the similar characteristics of the IDGC model (i.e., to achieve high classification accuracy and to stabilize imbalanced data sets) and maximizes the advantages of the IDGC model. The IDGC model introduces the AGC γ coefficient to deal with false classification of the minority class samples. The method strengthens the gravitational field of the minority class and reduces the false negative rate (regards the minority class samples as the negative class). The S-IDGC model modifies the weight coefficient optimization method and improves the efficiency of models.

5.3. S-IDGC based mobile malware detection

The performances of the IDGC and S-IDGC models on the mobile traffic data set are evaluated in this section. Time consumption is crucial in a real-time system to be deployed in real world. Similar to the experiments earlier, the IR value is set for 16 different groups and fivefold cross-validation is performed five times to obtain the evaluations in terms of AUC and GM. The SMOTE algorithm is utilized to resample data before training the IDGC model and the S-IDGC model, which are called SMOTE + IDGC and SMOTE + S-IDGC. The parameters of the SMOTE + IDGC and SMOTE + S-IDGC classifiers are shown in Table 9.

To evaluate the time consumption, the parameter of the maximum iteration time is set to 200 (as shown in Table 9:*MAX iteration time* = 200), which refers to the termination condition of the algorithm. All the experiments run on *Intel*(*R*) *Core*(*TM*)2 *Duo CPU E*7500, 2.93*GHz*, 4.0*G RAM*, 32*bit Windows* 7 computer. The AUC evaluation results of SMOTE + IDGC and the SMOTE + S-IDGC are shown in Fig. 9. Fig. 10 presents the GM evaluations of the two models. The results of the evaluation of time consumption are shown in Fig. 11.

From the AUC and GM evaluations in Figs. 9 and10, no significant difference exists in the classification performances of the improved S-IDGC algorithm and the original IDGC algorithm. Moreover, independent-sample *t*-test is conducted to

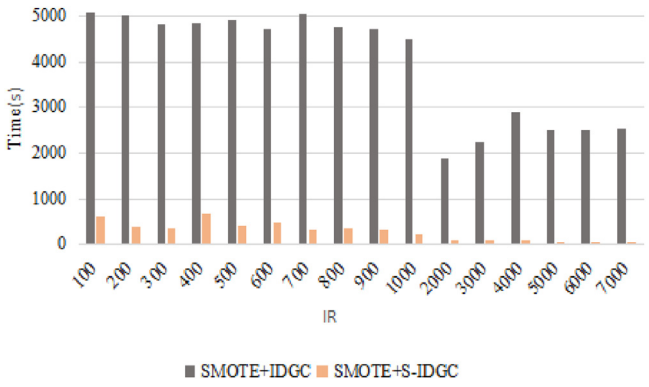


Fig. 11. Time evaluation results of SMOTE+IDGC and SMOTE+S-IDGC classifiers.

Table 10
AUC evaluation results of IDGC and S-IDGC classifiers on testing set.

	Levene test		T test						
	F	Sig.	t	df	Sig.(bilateral)	Mean error	Standard error	95% confidence interval	
								Upper bound	Lower bound
Equal variances assumed	0.365	0.550	−0.789	30	0.436	−0.01494	0.01894	−0.05362	0.02374
Equal variances not assumed			−0.789	29.906	0.437	−0.01494	0.01894	−0.05362	0.02375

Table 11
GM evaluation results of IDGC and S-IDGC classifiers on testing set.

	Levene test		T test						
	F	Sig.	t	df	Sig.(bilateral)	Mean error	Standard error	95% confidence interval	
								Upper bound	Lower bound
Equal variances assumed	0.383	0.541	−0.749	30	0.459	−0.01458	0.01945	−0.05429	0.02514
Equal variances not assumed			−0.749	29.903	0.459	−0.01458	0.01945	−0.05430	0.02515

evaluate whether a significant difference exists between the two methods. The *t*-test result in terms of AUC is shown in Table 10, and that in terms of GM is presented in Table 11. For the AUC Levene test, *Sig.* $\geq$ 0.05 refers to the homogeneity of variance. No significant difference exists among the variances. The assumption of equal variances is accepted. For the *t*-test result on the first line, no significant difference exists in the AUC evaluations between the two methods since *Sig.*(*bilateral*) $\geq$ 0.1. Similarly, Table 11 indicates that no significant difference exists on the GM evaluations between the two methods. Therefore, S-IDGC can achieve almost the same classification result as IDGC.

Recall that in Fig. 7(b), the AUC evaluation can reach over 0.8 on the testing sets of the current imbalanced classifiers when the IR value is less than 2000. However, when the IR value changes from 2000 to 7000, the results of the algorithms, such as SMOTE + C4.5, C4.5CS, and SVMCS, tend to decline, as shown in Fig. 7(b). When the IR value reaches 7000, the AUC evaluation results of the algorithms (SVMCS, C4.5CS, SMOTE + SVM, SMOTE + C4.5, BalanceCascade, and EasyEnsemble) are all less than 0.6. Fig. 9(b) indicates the AUC evaluation of SMOTE + IDGC and SMOTE + S-IDGC on the same testing set. When the IR value falls between 2000 and 7000, the AUC evaluation results of these two algorithms remain above 0.8.

Recall that Fig. 8(b) shows the GM evaluation result of current imbalanced classifiers on the testing set. When the IR values are less than 1000, various GM evaluations of the imbalanced algorithms are consistently above 0.85. However, when the IR values exceed 1000, the GM evaluation of the SMOTE + C4.5 algorithm declines significantly. When IR increases continuously, the GM evaluations of most algorithms begin to decline. When the IR value reaches 7000, the lowest GM evaluation is only approximately 0.3. However, the GM evaluations of SMOTE + IDGC and SMOTE + S-IDGC do not decrease when the value of IR increases. In fact, we find that AUC and GM evaluations tend to increase when the IR values are beyond 6000.

The stable AUC and GM evaluation results ensure that the detection result is minimally influenced by the degree of imbalance. The time-consumption performances of IDGC and S-IDGC are shown in Fig. 11. S-IDGC performs much better than IDGC. The time complexity of S-IDGC is approximately 17 times less than that of IDGC.

As the IR value continuously increases in a highly imbalanced environment, the IDGC model and the S-IDGC model perform stably. The efficiency of SMOTE + S-IDGC is better than that of SMOTE + IDGC. Based on the experimental results and the preceding analysis, we show the high stability and efficiency of the SMOTE + S-IDGC algorithm, which is suitable for malicious flow detection due to its high stability and efficiency under highly imbalanced conditions.

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044

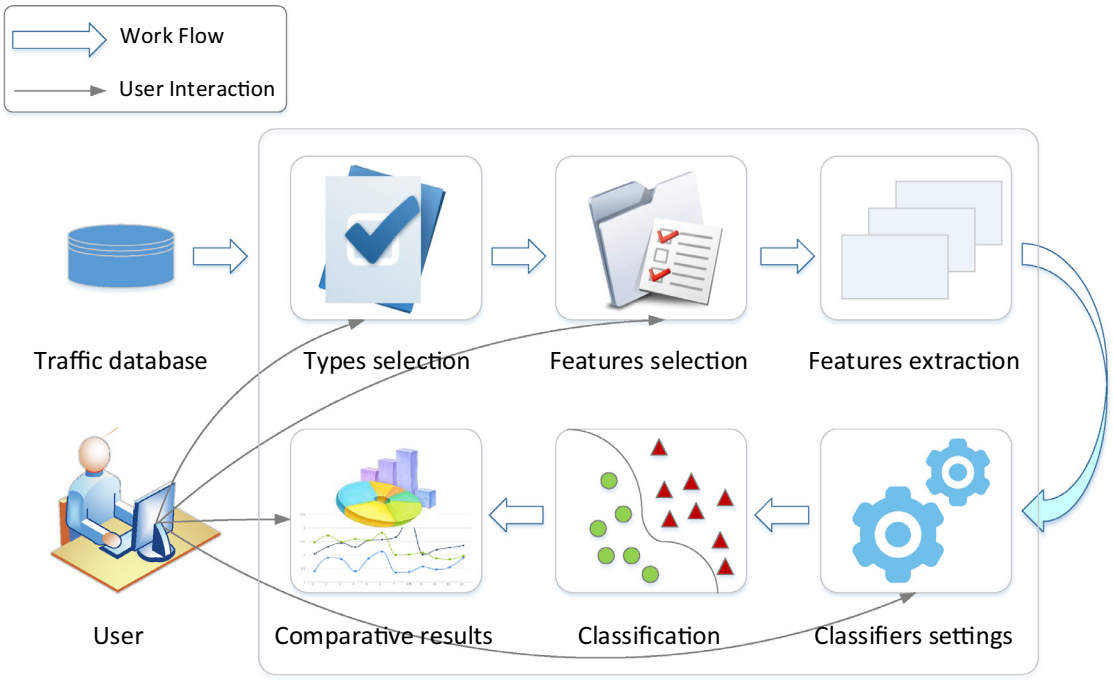


Fig. 12. A comparative benchmark prototype system framework for network traffic based malware detection.

6. Comparative benchmark prototype system

Comparing multiple traffic classifiers and quantifying their benefits remain a challenge nowadays. To solve this challenge, we design a comparative benchmark prototype system, which can help establish consensus among researchers on which approach to utilize and to determine when, where, and how an approach can be improved to identify a specific type of malicious app. The architectural overview of the prototype system is presented in Fig. 12. The prototype system has the following components: (1) unified network traffic data, (2) traffic processing and classifier settings, (3) comparative performance results, and (4) operating procedures of the system.

6.1. Unified network traffic data

One of the major problems in various studies on Android malware traffic is the lack of an effective and authoritative Android traffic data set. Some experts have investigated various classifiers. However, the comparative analysis has to be performed manually, since these methods all use different data sets. Thus, comparing these methods is difficult. To solve this problem, an automatic traffic acquisition platform, which addresses the problem of insufficient and non-uniform data, is designed. We suggest that this line of work should conduct comparisons on the same data set to make comparison results more reliable and convincing.

In Section 3.2, the automated traffic generation and collection platforms are introduced in detail. Substantial traffic data generated by malicious apps and benign apps can be obtained through this platform, which will be stored in the traffic database. This prototype system aims to compare the performance results of various classifiers in identifying malicious traffic. Thus, it is necessary to have a labeled traffic data set. The traffic data set in databases is divided into two sets: training set and testing set. The training set is used to train the classification models, whereas the testing set is used to test the performance of the models.

6.2. Traffic processing and classifier settings

This module includes traffic processing and classifier settings. The prototype system considers the diverse requirements of users and provides users with privileges, such as selecting a specific item. Users can select a specific type of traffic based on their requirements. For example, some users are concerned only about HTTP traffic. Other users focus on TCP flow, DNS traffic, and various types of traffic. Prototype systems can fulfill the specific requirements of users. After users select one type of traffic, the system automatically presents the related features of this type traffic to users. For example, users select TCP flow, and the system will show more selectable features related to TCP flow, such as the duration of flow and the number of bytes in an uplink traffic flow. From the selected features, the prototype system indicates that the corresponding

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044

function is required to extract these features. Users can add their own feature extraction function into the system to extract traffic features that are not included in the system.

After traffic processing, the classifiers and their parameters are configured. Users can choose comparative classifiers and configure their parameters. For standard classifiers, the ratio of the training set to the testing set and the fold number of cross-validation are considered. The IR values of the imbalanced classifiers can also be configured. Users can also add their own classifiers to this system. After these settings are complete, classification operations begin.

6.3. Comparative performance results

The same and appropriate evaluation criteria are used to assess the performance of different malicious traffic classification algorithm. To avoid the difficulties in evaluating performance, the prototype system uses multiple evaluation criteria to assess the performance of each classification algorithm. The utilized criteria include Accuracy Rate (ACC), Recall, F-Measure, Area Under Curve(AUC) and Geometric Mean (GM).

- $ACC = TP/(TP + FP)$. The complete descriptions of ACC, TP and FP are presented in Section 4.1.
- $Recall = TP/(TP + FN)$. Recall is the percentage of traffic with an app that is correctly identified.
- $F - Measure = 2 \times ACC \times Recall/(ACC + Recall)$. The F-Measure is a widely used metric in classification. The benchmark prototype system uses this metric to compare these classifiers and rank their performances.
- $AUC = (1 + TPR - FPR)/2$. The complete descriptions of AUC, TPR and FPR are provided in Section 4.1.2.
- $GM = \sqrt{TPR \times FPR}$. The complete descriptions of GM, TPR and FPR are provided in Section 4.1.2.

The benchmark prototype system can also evaluate the computational performance of malicious traffic classifiers by counting the classification time and training time. Classification time refers to that the time spent in classifying traffic data. Training time denotes the time spent on training classification models. However, the evaluation criteria of the current research are not only limited to these measures. Users can add other assessment measures depending on their specific requirements. The prototype system can visualize the result of multiple classifiers using histograms, pie charts, line charts, or other formats, to help users find appropriate classifiers under different scenarios.

6.4. Operating procedures of the system

The prototype system provides users with various knobs to configure and visualize the results of the experiment. Upon initiation, users can create a project, which is a basic operational unit. In each project, a user can select a series of classification algorithms for evaluation. Multiple features are used, and various evaluation criteria are adopted to describe the result. If users focus only on HTTP traffic, then an item of HTTP can be selected. Next, the system presents the features of the HTTP packet, which allows users to make a further choice. Subsequently, the means of visualization, such as a bar graph, pie chart, or line graph, is selected. Users can also adjust the weight of each classifier to obtain better classification results. After going through all the configuration steps, users can run this experiment by clicking the “run the project” button. The system presents the final comparative results based on user configurations. This prototype system allows users to easily obtain, compare, and visualize the classification result of different classifiers. By investigating the comparative results, the users are able to select most suitable classifiers to fulfill their specific needs.

7. Conclusion

In this paper, statistical features of mobile traffic are utilized to identify malicious traffic flows. After analyzing the traffic flow statistics, we discover the data imbalance issue that significantly affect the performance of Android malicious flow detection. Based on six network flow features extracted from the flow data set, we implement several classification algorithms, and run tests on different imbalanced data sets with different imbalance ratio (IR) values. The results show that most of the commonly-used classifiers achieve reasonable performance, which confirms that the machine learning algorithms are effective in identifying malicious mobile traffic. However, as the IR value increases, the accuracy of benign classifiers drops significantly with the increasing degree of imbalance. A comprehensive evaluation of the existing methods for the imbalanced problems shows that no existing methods can effectively address highly imbalanced problems when IR reaches higher than 4000.

We then evaluate the performance of IDGC model in addressing the data imbalance issue. After testing the IDGC model on the same traffic flow data set, we show that IDGC is significantly more stable than other classifiers. With increasing IR values, the classification performance of IDGC in terms of AUC and GM sustains in a range between 0.8 and 1.0. However, the classification process of IDGC is too time-consuming, which makes real-time detection impractical. To improve the efficiency of the IDGC model, we propose a novel S-IDGC model, which optimizes the weight coefficients using an efficient simplex method. The evaluation results show that S-IDGC inherits the stability characteristic of the IDGC model while drastically reduce the time consumption (with approximately 17 times improvement compared with IDGC on time efficiency). In addition, we design a comparative benchmark prototype system that integrates various types of machine learning classifiers for Android malicious traffic detection, which provides a neat service for users to have a fair comparison among different types of classifiers.

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044

Acknowledgement

This work was supported by the National Natural Science Foundation of China under Grants No. 61672262, No. 61573166, No. 61472164 and No. 61572230, the Natural Science Foundation of Shandong Province under Grants No. ZR2014JL042 and No. ZR2012FM010, the Shandong Provincial Key R&D Program under Grants No. 2016GGX101001. This work is also supported in part by NSF grant CNS-1566388.

References

[1] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, KEEL Data-mining software tool: data set repository, integration of algorithms and experimental analysis framework, J. Multi Valued Logic Soft Comput. 17 (2011) 255–287.
[2] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, Drebin: effective and explainable detection of android malware in your pocket, in: Proceedings of NDSS14, 2014.
[3] R. Barandela, J.S. Sánchez, V. García, E. Rangel, Strategies for learning in class imbalance problems, Pattern Recognit. 36 (3) (2003) 849–851.
[4] K. Bartos, M. Sofka, V. Franc, Optimized invariant representation of network traffic for detecting unseen malware variants, in: Proceedings of the 25th USENIX Security Symposium (USENIX Security 16), 2016, pp. 807–822.
[5] G.E. Batista, R.C. Prati, M.C. Monard, A study of the behavior of several methods for balancing machine learning training data, ACM Sigkdd Explor. Newslett. 6 (1) (2004) 20–29.
[6] A.P. Bradley, The use of the area under the ROC curve in the evaluation of machine learning algorithms, Pattern Recognit. 30 (7) (1997) 1145–1159.
[7] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, Smote: synthetic minority over-sampling technique, J. Artif. Intel. Res. 16 (1) (2011) 321–357.
[8] N.V. Chawla, A. Lazarevic, L.O. Hall, K.W. Bowyer, Smoteboost: improving prediction of the minority class in boosting, in: Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery, Springer, 2003, pp. 107–119.
[9] Z. Chen, H. Han, Q. Yan, B. Yang, L. Peng, L. Zhang, J. Li, A first look at android malware traffic in first few minutes, in: Proceedings of the 2015 IEEE Trustcom/BigDataSE/ISPA, 2015.
[10] M. Conti, L.V. Mancini, R. Spolaor, N.V. Verde, Analyzing android encrypted network traffic to identify user actions, IEEE Trans. Inf. Forensics Security 11 (1) (2016) 114–125.
[11] G.B. Dantzig, Application of the simplex method to a transportation problem, Activity Anal. Prod. Alloc. 13 (1951) 359–373.
[12] J.F. Díez-Pastor, J.J. Rodríguez, C.I. García-Osorio, L.I. Kuncheva, Diversity techniques improve the performance of the best imbalance learning ensembles, Inf. Sci. (Ny) 325 (2015) 98–117.
[13] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L.P. Cox, J. Jung, P. McDaniel, A.N. Sheth, Taintdroid: an informationflow tracking system for realtime privacy monitoring on smartphones, ACM Trans. Comput. Syst. (TOCS) 32 (2) (2014) 393–407.
[14] W. Enck, D. Octeau, P. McDaniel, S. Chaudhuri, A study of android application security, in: Proceedings of the USENIX Security Symposium, volume 2, 2011, pp. 1–2.
[15] A. Feizollah, N.B. Anuar, R. Salleh, F. Amalina, R.R. Maarof, S. Shamshirband, A study of machine learning classifiers for anomaly-based mobile botnet detection, Malaysian J. Comput. Sci. 26 (4) (2013) 251–265.
[16] A. Feizollah, N.B. Anuar, R. Salleh, A.W.A. Wahab, A review on feature selection in mobile malware detection, Digit. Invest. 13 (2015) 22–37.
[17] M. Galar, A. Fernández, E. Barrenechea, H. Bustince, F. Herrera, Ordering-based pruning for improving the performance of ensembles of classifiers in the framework of imbalanced datasets, Inf. Sci. (Ny) 354 (2016) 178–196.
[18] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The weka data mining software: an update, ACM SIGKDD Explor. Newslett. 11 (1) (2009) 10–18.
[19] D. He, S. Chan, M. Guizani, Mobile application security: malware threats and defenses, Wireless Commun. IEEE 22 (1) (2015) 138–144.
[20] H. He, E.A. Garcia, Learning from imbalanced data, IEEE Trans. Knowl. Data Eng. 21 (9) (2009) 1263–1284.
[21] Y. Jin, N. Duffield, J. Erman, S.S. Haffner P. and, Z.-L. Zhang, A modular machine learning system for flow-level traffic classification in large networks, ACM Trans. Knowl. Discov. Data (TKDD) 6 (1) (2012) 1–4.
[22] C. Lever, M. Antonakakis, B. Reaves, P. Traynor, W. Lee, The core of the matter: analyzing malicious traffic in cellular carriers, in: Proceedings of the NDSS13, 2013.
[23] S. Liu, Y. Wang, J. Zhang, C. Chen, Y. Xiang, Addressing the class imbalance problem in twitter spam detection using ensemble learning, Comput. Security (2016).
[24] X.Y. Liu, J. Wu, Z.H. Zhou, Exploratory undersampling for class-imbalance learning, IEEE Trans. Syst. Man Cybern. Part B Cybern. 39 (2) (2009) 539–550.
[25] Z. Liu, R. Wang, M. Tao, X. Cai, A class-oriented feature selection approach for multi-class imbalanced network traffic datasets based on local and global metrics fusion, Neurocomputing 168 (2015) 365–381.
[26] V. López, A. Fernández, F. Herrera, On the importance of the validation technique for classification with imbalanced datasets: addressing covariate shift when data is skewed, Inf. Sci. (Ny) 257 (2014) 1–13.
[27] V. López, A. Fernández, S. García, V. Palade, F. Herrera, An insight into classification with imbalanced data: empirical results and current trends on using data intrinsic characteristics, Inf. Sci. (Ny) 250 (2013) 113–141.
[28] M. Marchetti, F. Pierazzi, M. Colajanni, A. Guido, Analysis of high volumes of network traffic for advanced persistent threat detection, Comput. Netw. 109 (2016) 127–141.
[29] F.A. Narudin, A. Feizollah, N.B. Anuar, A. Gani, Evaluation of machine learning classifiers for mobile malware detection, Soft Comput. 20 (1) (2016) 1–15.
[30] W.W. Ng, G. Zeng, J. Zhang, D.S. Yeung, W. Pedrycz, Dual autoencoders features for imbalance classification problem, Pattern Recognit. 60 (2016) 875–889.
[31] L. Peng, B. Yang, Y. Chen, A. Abraham, Data gravitation based classification, Inf. Sci. (Ny) 179 (6) (2009) 809–819.
[32] L. Peng, H. Zhang, Y. Chen, B. Yang, Imbalanced traffic identification using an imbalanced data gravitation-based classification model, Comput. Commun. (2016) 1–13.
[33] L. Peng, H. Zhang, B. Yang, Y. Chen, A new approach for imbalanced data classification based on data gravitation, Inf. Sci. (Ny) 288 (2014) 347–373.
[34] R. Perdisci, W. Lee, N. Feamster, Behavioral clustering of http-based malware and signature generation using malicious network traces, in: Proceedings of the NSDI10, 2010, pp. 391–404.
[35] J.A. Sáez, J. Luengo, J. Stefanowski, F. Herrera, Smotecipf: addressing the noisy and borderline examples problem in imbalanced classification by a re-sampling method with filtering, Inf. Sci. (Ny) 291 (2015) 184–203.
[36] A. Shabtai, L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, Y. Elovici, Mobile malware detection through analysis of deviations in application network behavior, Comput. Security 43 (2014) 1–18.
[37] M. Stevanovic, J.M. Pedersen, Machine Learning for Identifying Botnet Network Traffic, Tech. Rep., 2013. http://vbn.aau.dk/files/75720938/paper.pdf.
[38] K.M. Ting, An instance-weighting method to induce cost-sensitive trees, IEEE Trans. Knowl. Data Eng. 14 (3) (2002) 659–665.
[39] K. Veropoulos, C. Campbell, N. Cristianini, et al., Controlling the sensitivity of support vector machines, in: Proceedings of the International Joint Conference on AI, 1999, pp. 55–60.
[40] VirusTotal, URL: http://virustotal.com/.
[41] S. Wang, Z. Chen, L. Zhang, Q. Yan, B. Yang, L. Peng, Z. Jia, Trafficav: an effective and explainable detection of mobile malware behavior using network traffic, in: Proceedings of the 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS), 2016, pp. 1–6.

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044

[42] K. Xu, Y. Li, R.H. Deng, Iccdetector: icc-based malware detection on android, IEEE Trans. Inf. Forensics Security 11 (6) (2016) 1252–1264.
[43] H. Zhang, W. Banick, D. Yao, N. Ramakrishnan, User intention-based traffic dependence analysis for anomaly detection, in: Proceedings of the 2012 IEEE Symposium on Security and Privacy Workshops (SPW), IEEE, 2012, pp. 104–112.
[44] H. Zhang, D. Yao, N. Ramakrishnan, Z. Zhang, Causality reasoning about network events for detecting stealthy malware activities 1, Comput. Security 58 (C) (2016) 180–198.
[45] J. Zhang, H. Li, Q. Gao, H. Wang, Y. Luo, Detecting anomalies from big network traffic data using an adaptive detection approach, Inf. Sci. (Ny) 318 (C) (2015) 91–110.
[46] Y. Zhou, X. Jiang, Dissecting android malware: characterization and evolution, in: Proceedings of the 2012 IEEE Symposium on Security and Privacy, 2012, pp. 95–109.

Please cite this article as: Z. Chen et al., Machine learning based mobile malware detection using highly imbalanced network traffic, Information Sciences (2017), http://dx.doi.org/10.1016/j.ins.2017.04.044