Computers & Security 137 (2024) 103604

Contents lists available at ScienceDirect

Computers & Security

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

SynDroid: An adaptive enhanced Android malware classification method based on CTGAN-SVM

Junhao Li [a], Junjiang He [a,*], Wenshan Li [a], Wenbo Fang [a], Geying Yang [b], Tao Li [a]

[a] School of Cyber Science and Engineering, Sichuan University, China
[b] School of Cyber Science and Engineering, Wuhan University, China

ARTICLE INFO

Keywords:
Android malware classification
GAN
Imbalanced dataset
Data augmentation

ABSTRACT

Android mobile phones have the highest market share nowadays, bringing a boom of Android application programming as well as malicious software (malware) issues. Traditional machine learning and deep learning methods are widely used in Android malware detection and classification, both require plenty of application samples to train classifiers. However, the collected samples are always imbalanced, because the distribution of malware categories differs hugely in the real environment. For Android malware samples with high dimension features, and Class Imbalance Ratio coming to more than 100:1, traditional methods become weak. To fill the gap, we propose an Android malware classification model named SynDroid. The core step of SynDroid uses CTGAN-SVM to generate qualified high-dimension samples, and adaptively discards bad results. Besides, we propose KS-CIR test to help the model to determine which classes of data needed to be enhanced most. This new proposed test can measure the data in respect of both samples' quality and quantity. Lastly, Random Forest is taken as a classifier to finish the classification task. The performance of SynDroid is evaluated on CCCS-CIC-AndMal2020 on the accuracy, precision, recall and F1-score. Both longitudinal and horizontal comparison experiments have been done with traditional oversampling methods, cost-sensitive learning, and other complicated methods. The result shows that the proposed method gets 12% more accuracy than the method on the same dataset and alleviates imbalanced data problems.

1. Introduction

According to the statistics, Android-OS-based mobile phones account for 70.79% of the whole smartphone market in 2022's first quarter (Smartphone market, 2022). The highest market share has brought a boom of Android application programming, as well as Android malicious software (malware) issues. As the fact that modern people engage most of their economic activities through mobile phones, hackers can embed malware to steal users' sensitive information such as bank transaction records, personal passwords, ID numbers, social relationships, and so on. For example (Goldoson, 2023), a new Android malware named 'Goldoson' has infiltrated Google Play through 60 legitimate apps that have over 100 million downloads. It can illegally gather sensitive data and inject advertisement functions, but the victim can not see any indication of these activities on their device. It is crucial to develop Android malware detection and classification technology to deal with these problems.

In the past decade, researchers have tried to utilize plenty of machine learning (ML) algorithms to defend against malware. For example, Aafer et al. (2013) conduct a thorough analysis to extract relevant features to malware behavior captured at the API level. Then they use the K-Nearest-Neighbors algorithm as the classifier and get the best result. Arp et al. (2014) propose Drebin, a lightweight method using SVM to detect Android malware. Drebin is effective and enables identifying malicious applications directly on the smartphone. Fereidooni et al. (2016) propose ANASTASIA, a system using XGboost to detect Android malware through a large number of static analyses. These methods commonly analyze features of Android permissions and API calls to distinguish whether an application is a malware (Aafer et al., 2013; Wu et al., 2021). Lately, with the trend of deep learning, researchers have applied deep neural networks to detect malware and achieved promising results. For instance, Feng et al. (2019) propose MobiDroid, leveraging CNN as the detector to provide a real-time secure and fast response environment on Android devices. To figure out the potential detection accuracy promotion of different deep neural networks, R. Feng et al.

* Corresponding author.
E-mail address: hejunjiang@scu.edu.cn (J. He).

https://doi.org/10.1016/j.cose.2023.103604
Received 26 July 2023; Received in revised form 9 October 2023; Accepted 15 November 2023
Available online 22 November 2023

not only apply extracted features with CNN models but also present recurrent neural network models (e.g., LSTM and GRU (Feng et al., 2020, 2021)). All these methods faced the same problem that the distribution of Android malware categories differs hugely (Android malwares, 2021), so the collected datasets are always imbalanced.

Generative Adversarial Network (GAN) (Goodfellow et al., 2014) models are massively used in Computer View research. GAN models can synthesize qualified samples despite the high dimension features or great imbalance ratio, which are both typical characteristics of Android malware. Thus, now researchers use this technology in malware detection and classification, aiming to increase the diversity of minority data. Renjith et al. (2021) use GAN directly to generate Android malware samples in tabular expression to expand the training dataset. Kim et al. (2018) use GAN to generate fake Android samples. The fake data has some new features, so the detector can learn from it and discover some zero-day malware. Chen et al. (2021) use GAN to generate malware samples in image expression, then fit the data into the CNN model and get better accuracy.

To sum up, GAN models can effectively improve the imbalanced dataset, and enhance the detection or classification result. However, it is common that even after plenty of iterations, the GAN model is probably still not convergent or even collapses (Chen, 2021). Researchers should manually check the generated data and tune the GAN model in time. But it is difficult and time-consuming to check the tabular data, which cannot be distinguished and discarded with just a glimpse like image data. Besides, there are many traditional methods dealing with imbalanced problems. Cost-sensitive learning (Zhou and Liu, 2006) tunes learning weight according to the cost matrix, aiming to let the model pay more attention to the minority class. Data-level approaches like oversampling (Chawla et al., 2002; He et al., 2008; Hui et al., 2005; Last et al., 2017) directly alter the data distribution to increase the amount of minority data. These methods merely focus on the quantity gap between minority and majority classes, but they neglect the samples' quality gap. Some malware is unique enough that models can easily distinguish them from others, despite the minor proportion. Some malware though has a larger proportion, but it still needs to be enhanced because the features are not prominent. Therefore, classification results are based not only on the samples' quantity but also on the samples' quality.

To fill the mentioned gap, an Android malware classification framework named SynDroid (using CTGAN-SVM to **syn**thesize An**droid** malware) is proposed. We found that SVM can truly replace human checking while training a GAN model. It can promptly be aware of the deviation of generated data and properly feedback to the GAN model. Then the parameters of GAN are tuned and the generated data will be more qualified. We also propose a novel evaluation method named KS-CIR test to measure both samples' quality and quantity. We experiment SynDroid at CCCS-CIC-AndMal2020, a big but imbalanced dataset consisting of over 200K Android malware of 14 families. We find that SynDroid gets the best performance after a series of longitudinal and horizontal experiments. In summary, the contributions of this paper are as followed:

- **A novel evaluating method before data augmentation**: A brand new evaluating method named KS-CIR test is proposed. We combine the Kolmogorov-Smirnov test and Class Imbalance Ratio to measure both samples' quality and quantity, therefore determining whether a minority class needs to be augmented. It performs better than cost-sensitive learning or the traditional over-sampling methods which merely focus on the quantity gap.
- **Use CTGAN-SVM to adaptively alleviate the imbalance problem**: We implement CTGAN (Lei and Veeramachaneni, 2018) to synthesize Android malware samples, the samples are more qualified than traditional over-sampling methods. Moreover, our method can automatically discard bad results and tune the model's parameters properly. SVM replaces the role that researchers must play to manually check the results and the process is less laborious.
- **Sufficient verification of the proposed method**: Both longitudinal and horizontal comparison experiments have been done on CCCS-CIC-AndMal2020. AdnSyn gets better performance in terms of Accuracy, Recall, Precision, and F1-score compared with other methods.

The rest of the paper is organized as follows. *Background and related work* section presents background knowledge and previous related work. *Proposed method* section describes the details of the SynDroid model and KS-CIR test. Experimental results and discussion are given in *Experiments and discussion* section. Conclusion and future work are in *Conclusion and future work* section. *Acknowledgment* section expresses the support of the paper.

2. Background and related work

2.1. Android malware analysis

According to different ways of processing malware samples, malware analysis methods can be divided into static analysis and dynamic analysis.

2.1.1. Static analysis

The static analysis aims to extract features without executing or installing the sample. Due to its peculiarity, the detector can get excellent detection results in a short time if the samples are from known malware families. But this method performs not well when samples use obfuscated technology or come from a brand new malware category. In the Android malware detection context, the static analysis commonly extracts code information, API calls, or permissions as samples' typical characteristics.

To analyze code information, Chen et al. (2015) proposed a binary classification malware detection system named TiniDroid, using Apktool to decompile the APK file into Smali code which is a more explainable code form compared with Dalvik bytecode. Then the system computes the n-grams of the Smali code and uses the n-grams result as the basis of classification. Liu et al. (2021) were inspired by the fact that malware from the same family always shares a similar code and proposed NSDroid. NSDroid detects malware according to the apps' function call graphs.

To analyze API calls and permissions, Suarez-Tangil et al. (2017) proposed DroidSieve. DroidSieve extracts static features including API calls occurring in the code, the request permission, and other app components. After a comprehensive inspection of these characteristics, the data is fed into several classification models including SVM, Extra Trees and XGBoost, getting a detection of 99.44%. Qiao et al. (2016) proposed a malware-detecting method to extract the API calls and permissions automatically. The authors decompile **.dex* bytecode to java code and create a list of API calls, which is found more accurate than examining the *AndroidManifest.xml* file to obtain the permissions. Wu et al. (2012) proposed DroidMat not only focusing on the API calls themselves, but also the service and activity called by the API. The reason for obtaining extra information is that even the same API calls may have different intentions according to the different components they are arranged in.

2.1.2. Dynamic analysis

Dynamic analysis needs to run the samples, then get their real behavior and their influence on the running environment. It performs well even if we have no prior knowledge about the malware, but consumes more time and detects a threat just the moment it is about to occur. Dynamic analysis monitors specific behaviors including system calls, other system-level behavior and user-space-level behavior.

To monitor system calls, Xiao et al. (2018) utilized the Long Short-Term Memory model as the classifier and took the whole sequence of a given call into consideration. In their method, a system call is seen as an

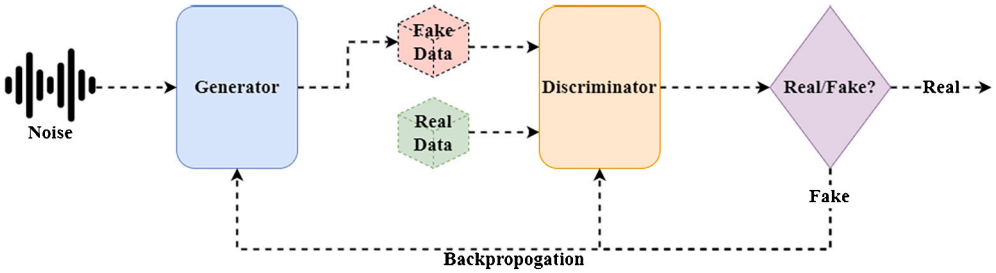


Fig. 1. The structure of GAN.

independent "word", thus the whole sequence of system calls is considered as a natural language sentence. Canfora et al. (2015) also proposed a detection method based on system calls. The proposed method can select important ones for the detection result from plenty of system calls, finding that malware tends to have some common behaviors, and the same behavior can be encoded in a semantically identical way.

Besides system calls, there is other system-level information that can sign malware. Feng et al. (2018) proposed EnDroid, extracting dynamic behavior characteristics including network operation, cryptographic operation, file operation, telephone calls, SMS messages, receiver actions, receiver startup, **.dex* file loading, information leaks, and system calls. These ten characteristics are converted into feature vectors, the classifiers detect benign and malicious samples according to these vectors and get a detection accuracy of 97%.

User-space level information refers to API calls rather than system-level calls. To utilize this kind of information, Xie et al. (2018) proposed RepassDroid combining semantic and syntactic analyzing to detect malware. RepassDroid's feature extraction module regards the API calls as a semantic function and the essential permissions as a syntactic function, forming them into feature vectors then putting the vectors into the classifier module, and getting the accuracy of 97.7%.

2.2. CTGAN

Generative Adversarial Networks originated in Goodfellow et al. (2014), its architecture includes two independent neural networks: Generator and Discriminator, which are shown in Fig. 1. The generator as its name suggests, is to generate fake data from noise data, and the discriminator judges the fake data after learning from the real data. The repeated iterative training procedure lastly makes the generated data completely similar to real data. CTGAN (Lei and Veeramachaneni, 2018) is especially proposed to generate data in tabular form. Complexities come for CTGAN because of the diversity of data in the table, including numerical, categorical, time, text, and cross-table references. Thus, CTGAN pre-processes the data in reversible ways which are as followed:

- **Mode-specific normalization for numerical variables**. CTGAN uses a Gaussian Mixture model (GMM) to cluster values of a numerical variable to overcome the bad result by simply normalizing numerical features to [-1, 1] or just using tanh activation to generate the features.
- **Smoothing for categorical variables.** CTGAN converts categorical variables to one-hot-encoding representation and adds noise to binary variables, then generates the probability distribution directly using softmax.

CTGAN takes long-short-term memory (LSTM) network as the generator and uses Multi-Layer Perceptron (MLP) in the discriminator. Especially, to warm up the model more efficiently, CTGAN jointly optimizes the KL divergence of discrete variables and the cluster vector of continuous variables by adding them to the loss function. Adding the KL divergence term can also make the model more stable.

2.3. Data augmentation in malware detection

Data augmentation used in data science aims to increase the amount of data by modifying the existing data slightly or creating brand new data according to the existing data. It can expand, balance the dataset and reduce overfitting while training (Data augmentation, 2023). Traditionally, it is closed to over-sampling methods (Chawla et al., 2002; He et al., 2008; Hui et al., 2005; Last et al., 2017) and is often used to process image data. Data augment applied in malware is promising research, it can be seen from two angles.

For the attackers, they try to use data augment technology to create adversarial malware samples to bypass the detector (which is also known as Adversarial Learning) (Rosenberg et al., 2021). For example, Hu and Tan (2017) uses GAN to create malware samples to perform black-box attacks, decreases the detection rate to nearly zero, and makes the retraining-based defensive method against adversarial examples hard to work. Chen et al. (2019) uses the strategy that gradually modifying the samples' features to find the decisive gap between benign samples and malware. In that way, the attackers can change the key features of the samples to mislead the CNN classifier.

For the defender, the goal is to use data augment technology to generate more malware samples to improve their detecting model. For example, Kim et al. (2018) uses GAN to generate fake data from a random distribution, it is similar to the real data and includes some modified features. The fake data can be assumed as novel malware, so the detector can learn from it and discover some zero-day malware. Wang et al. (2020) uses adversarial samples to enhance the robustness of their GCNN model, which is proposed to unearth the prediction sequence based on the system-call sequence augmentation method. Chen et al. (2018) aims to find the classifying boundary of the detector by converting the program into vector expressions, then transforming the binary feature space of samples into continuous probabilities. Vasan et al. (2020) traditionally uses image amplification methods such as rotation angle, offset, scaling, etc. to change the code images. Chen et al. (2021) uses GAN to augment the malware samples which are processed into image expression, then fit the data into the CNN model, finding that the detection result is improved after data augmentation. Renjith et al. (2021) uses GAN directly to generate the malware samples in tabular expression to expand the training dataset, but does not evaluate the synthetic data's quality and does not test the detection result.

3. Proposed method

We proposed a malware classification framework named SynDroid to synthesize samples that are in less portion or do not have prominent features, the architecture of SynDroid is shown in Fig. 2.

SynDroid consists of three main steps. The first step is to use a novel method named KS-CIR test to check the imbalanced dataset. KS-CIR test figures out which classes need to be enhanced, taking both samples' quality and quantity into consideration. The second step is to apply CTGAN-SVM to synthesize malware samples of certain classes according to the KS-CIR test result. The third step is to add the highly qualified generated samples to the dataset and to use Random Forest to complete

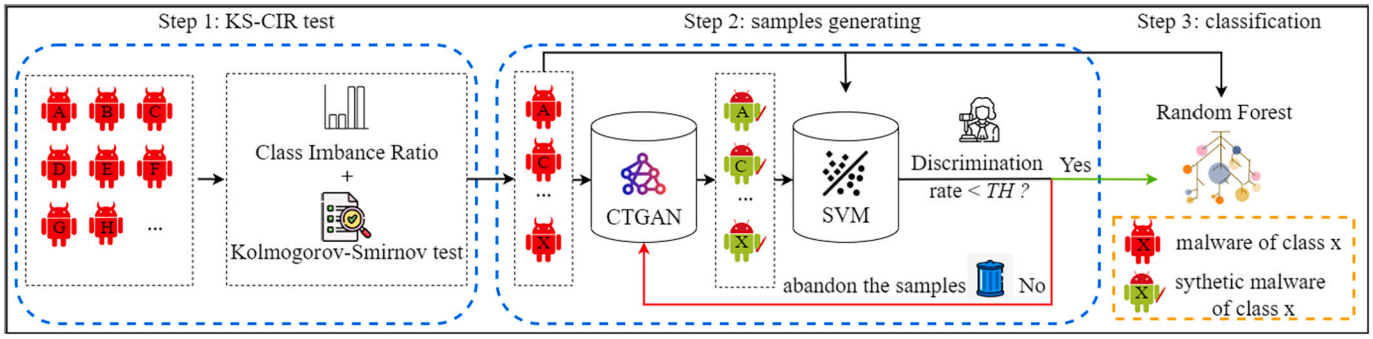


Fig. 2. The structure of SynDroid.

the malware classification task. The details of the architecture will be discussed in flowing sub-sections.

3.1. Step 1 of SynDroid

In a malware classification task, the classifier will perform badly if the samples are insufficient (quantity problem) or do not have prominent features (quality problem). Hence, the first step of SynDroid is the KS-CIR test, taking both quantity and quality into account, aiming to figure out which classes in the dataset needed to be enhanced.

High quality of samples means the samples are unique enough and a classifier can easily distinguish them from others. Thus, the Kolmogorov-Smirnov test in the KS-CIR test here is to measure the uniqueness of samples. Two-sample K-S test is one of the most useful and general non-parametric methods for comparing two samples, as it is sensitive to differences in both location and shape of the empirical cumulative distribution functions of the two samples. It actually answers the question 'What is the probability that these two sets of samples were drawn from the same (but unknown) probability distribution?' (Kolmogorov-Smirnov test, 2023). For a group of data $[x_1, x_2, x_3..x_n]$, the empirical distribution function is defined in Eq. (1)

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} 1_{[-\infty,x]}\left(X_i\right) \tag{1}$$

where $1_{[-\infty,x]}\left(X_i\right)$ is an indicator function, if $X_i < x$ it equals 1 otherwise it equals 0. The K–S statistic for two given empirical distribution function $F_{1,n}(x)$, $F_{2,m}(x)$ is defined in Eq. (2)

$$D_{n,m} = sup_x \left|F_{1,n}(x) - F_{2,m}(x)\right| \tag{2}$$

where sup_x is the supremum of the sets of distances. Intuitively, the Kolmogorov–Smirnov statistic takes the largest absolute difference between two the empirical distribution function $F_{1,n}(x)$, $F_{2,m}(x)$ (see Fig. 3).

As it is shown in Fig. 5, class A consists of N samples and class B consists of M samples, both with n features $[F_1, F_2, F_3 \ldots F_n]$, the Kolmogorov–Smirnov statistic D of F_i is calculated with Eq. (2). After calculating the Kolmogorov–Smirnov statistic D of every feature, we take the average of D as the K-S score S between class A and class B, which is given in Eq. (3)

$$S = \sum_{i=1}^{n} D_{F_i}/n \tag{3}$$

where n is the number of features. The purpose of K-S test is to measure the uniqueness of every data class. Once we use Eq. (3), we compare the difference between two classes. But the K-S test cannot stop at just two-class level. We have to further apply K-S score to every class level. Then we calculate the K-S score S of every pair of classes in the dataset, the detail is given in Algorithm 1.

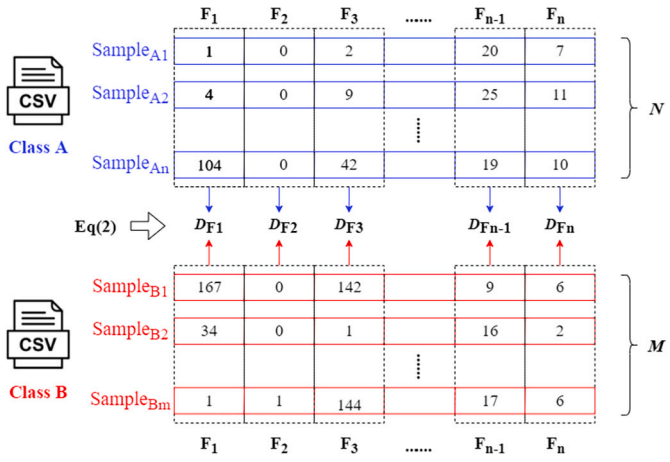


Fig. 3. The detail of K-S test.

Algorithm 1 Calculate S $(X_1[F_1(M), F_2(M)...F_n(M)], X_2[F_1(N), F_2(N)...F_n(N)])$.

Input: M samples of Class X_1 including n features, N samples of Class X_2 including n features
Output: K-S test score S of Class X_1 and Class X_2
1: $D_{sum} = 0$
2: **for** each F_i in X **do**
3: compute EDF(F_i of X_1) according to Eq. (1)
4: compute EDF(F_i of X_2) according to Eq. (1)
5: D_{F_i} = |EDF(F_i of X_1) - EDF(F_i of X_2)|
6: $D_{sum} += D_{F_i}$
7: $S = D_{sum}/n$
8: **return** S

The final K-S test result KS_i of a certain class i is defined in Eq. (4)

$$KS_i = 1 - \frac{\sum_{j=1, j\neq i}^{c} S_{i,j}}{c-1} \tag{4}$$

where c is the number of classes in the dataset. KS_i finally measures the uniqueness of class i and will be sorted from largest (least unique) to smallest (most unique).

The class imbalance ratio in the KS-CIR test is used to measure the imbalance degree of the data. CIR of class i is defined in Eq. (5)

$$CIR = M/m_i \tag{5}$$

where M is the amount of the majority-class samples and m_i is the amount of the minority-class samples. CIR will be calculated in every class and also sorted from largest to smallest.

Now we get two ordered lists consisting of classes' names according to the consisting score and CIR separately, then we need to select alternative classes from both lists. The selecting rule of the K-S list is defined in Eq. (6)

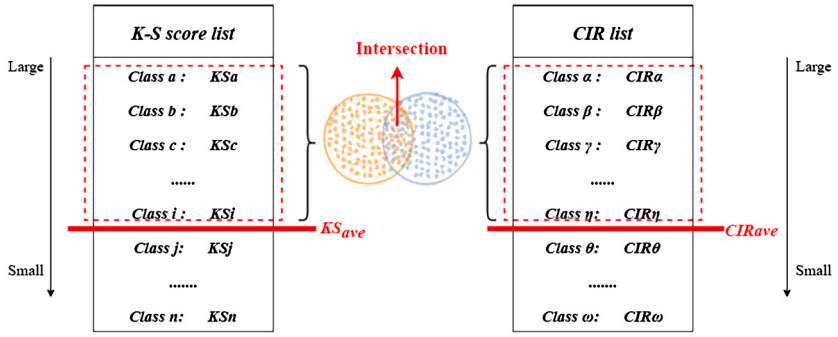
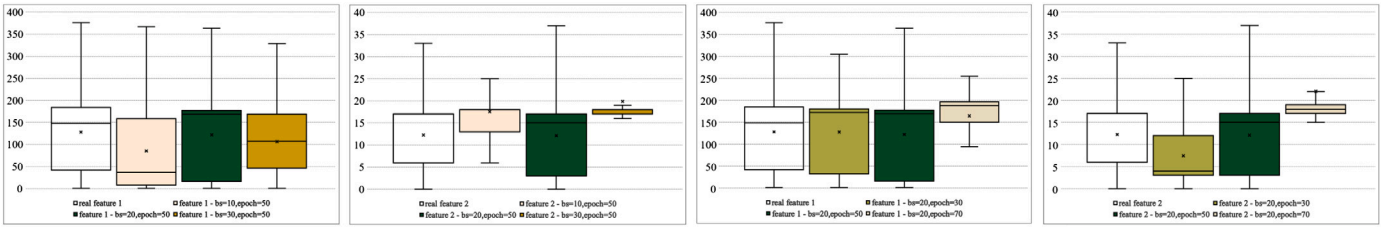


Fig. 4. The result of K-S CIR test.



feature 1, epoch = 50 feature 2, epoch = 50 feature 1, batch size = 20 feature 2, batch size = 20

Fig. 5. Synthetic samples with different parameters.

$$A = \{ i \,|\, KS_i > KS_{ave} \} \tag{6}$$

where KS_i is the K-S test result of class i calculated with Eq. (4) and KS_{ave} is the average of all K-S test results. If a class with a KS score is under the average, it is considered not prominent and needed to be enhanced. The selecting rule of the CIR list is defined in Eq. (7)

$$B = \{ i \,|\, CIR_i > CIR_{ave} \} \tag{7}$$

where CIR_i is the CIR of class i calculated with Eq. (5). In this paper, we decide the malware classes that have CIR above average are the minority class needed to be enhanced.

Finally, as it is shown in Fig. 4, we take the intersection of two sets as the result of the KS-CIR test. Classes included are the ones needed to be enhanced. The whole step of calculating the final KS-CIR test result is given in Algorithm 2.

Algorithm 2 KS-CIR test ($X_1[\alpha]$[n],$X_2[\beta]$[n]...$X_c[\psi]$[n]).

Input: α samples of Class X_1, β samples of Class X_2...ψ samples of Class X_c including n features
Output: a list of Classes that needed to be enhanced

$S[c]$: the summon of S of every two class
$KS[c]$: K-S test score of every class
$CIR[c]$: Class Imbalance Ratio of every class
$KS-CIR[x]$: list consists of x classes needed to be enhanced
M = MAX(α,β...λ)
for each Class i **do**
  for each Class j except Class i **do**
    compute $S_{i,j}(X_i,X_j)$ according to Algorithm 1
    $S[i] += S_{i,j}(X_i,X_j)$
  $KS[i] = S[i]/(c-1)$
  $CIR[i]$ = the amount of class i / M
$CIR_{ave} = Average(CIR[c])$
$KS_{ave} = Average(KS[c])$
x = 0
for $KS[i] > KS_{ave}$ **do**
  for $CIR[j] > CIR_{ave}$ **do**
    if $i == j$ **then**
      $KS-CIR[x] = i$
      $x++$
return $KS-CIR[x]$

3.2. Step 2 of SynDroid

The second step of SynDroid is to synthesize the samples and check the quality of the synthesized samples. After the KS-CIR test, the enhancing classes are determined and these samples are about to send to CTGAN.

The amount of the generating samples is a hyperparameter. In the traditional over-sampling method, the number of minority classes is simply increased equally to the majority class. But for datasets of Android malware samples with hundreds of features, Class Imbalanced Ratio varies from 10 to 100, this simple processing does not perform well. So in this paper, we define the baseline amount in Eq. (8)

$$N = \frac{M}{CIR_{ave}} \tag{8}$$

where M is the amount of the majority-class samples and $CIRave$ is the average of CIR. We also use the traditional baseline amount with the over-sampling method, experiments result will be given in Section 4 and shows that our method performs better.

The GAN model is usually used to generate image data. In malware classification scenario, transforming the samples into image form may be a novel idea (Jian et al., 2021), but that will lose the interpretability of detecting result (we cannot tell what a pixel actually means in a malware image), and transforming samples to image form also consumes plenty of time. Hence in most cases, application samples are processed into feature vectors, so we choose Tabular GAN, a GAN model that can directly generate data in tabular form.

There is a common problem for GAN that even after plenty of iteration, the model probably still generates low-quality data. This situation happens when the loss function of the GAN comes to a threshold, and the discriminator cannot correctly distinguish the generated data from the real data. So it is necessary for us to manually check the generated data, and opportunely tune the parameter inside the model to avoid bad results. But the situation gets complicated when the amount of generated data is huge, and the tabular data unlike image data cannot be distinguished as a bad result with just a glimpse.

Therefore, it is more efficient to set a supervisor outside the GAN model instead of manually checking. SVM is a simple but effective model which is massively used in binary classification. The character-

istics of SVM can serve our demand that the combined CTGAN-SVM model is not very complicated and can adaptively generate qualified data. As shown in Fig. 2, SVM actually plays the role of judge to ensure this self-adaptive process. If the SVM's discrimination rate is under a threshold ***TH***, it can be considered that the synthetic samples have high quality and therefore adopt them. Otherwise, we abandon them and tune the parameter inside the model then restart it. After testing, ***TH*** determined as 5% get good results. The detail of step 2 is shown in Algorithm 3. An example of the self-adaptive procession with the guide of SVM will be given in Section 4.

Algorithm 3 Generator ($X_1[A][\mathrm{n}]$, $X_2[B][\mathrm{n}]...X_x[\Psi][\mathrm{n}]$).

Input: A samples of Class X_1, B samples of Class $X_2...\Psi$ samples of Class X_x including n features

Output: α synthetic samples of Class Y_1, β synthetic samples of Class $Y_2...\psi$ synthetic samples of Class Y_x including n features

1: **for** each Class X_i **do**
2: **repeat**
3: $Y_i = \mathrm{CTGAN}(X_i)$
4: **until** accuracy of SVM(X_i,Y_i) < threshold TH
5: **return** $Y_1[\alpha][\mathrm{n}]$, $Y_2[\beta][\mathrm{n}]...Y_x[\psi][\mathrm{n}]$

3.3. Step 3 of SynDroid

The third step of SynDroid uses the Random Forest classifier to complete the Android malware classification task, the classifier receives both original malware samples and synthetic samples. Random Forest is an ensemble learning method, whose result is determined by the most tree classifiers inside RF. It is widely used in classification and regression tasks since it is first proposed in 1995 (Ho, 1995). Android malware classification essentially is a yes-or-no problem which is very proper for a tree model with a branching structure, that's why we choose Random Forest as the classifier of our method even though it is not very novel. We also compare RF with other classifiers to verify the advantages of our method, the horizontal and longitudinal experiments will be given in Section 4. The result shows that although Random Forest is less advanced than near-proposed deep learning models, it performs better with our SynDroid method.

4. Experiments and discussion

4.1. Experimental setup

Dataset: CCCS-CIC-AndMal-2020 (Keyes et al., 2021; Rahali et al., 2020) is a big dataset consisting of over 200K Android malware from 14 categories, collected by the Canadian Institute for Cybersecurity (CIC) in collaboration with the Canadian Centre for Cyber Security (CCCS). All the samples are processed with reserve engineering of *AndroidManifest.xml* file, and the extracted features include:

1. Activities
2. Broadcast receivers and providers
3. Metadata
4. The permissions requested by the application
5. System features (such as camera and Internet)

Besides, we apply the Gini Index (Gini Impurity) to select features. The Gini Index can calculate the contribution of each feature to the classification result. It is defined as

$$Gini(p) = \sum_{k=1}^{K} p_k \left(1 - p_k\right) \tag{9}$$

where there are K categories. The probability of a sample belonging to category k is p_k. If the data set D is divided into subset D_1, $D_2...D_m$ according to different values of feature A, the Gini Index of feature A is defined as

$$Gini(D;A) = \sum_{i=1}^{m} \frac{|D_i|}{|D|} Gini\left(D_i\right) \tag{10}$$

After calculating the Gini Index of every feature, we select 220 of them according to sorting. All the features are in integer format, including two kinds of data, 0-1 value and continuous value. The 0-1 value means that malware do an activity (e.g. apply for permission) or not. The continuous value means that the number of malware do the same activity (e.g. times of calling the camera). Either discrete 0-1 value or continuous value can be well calculated with Eq. (1) and Eq. (2). Thus, the K-S test can be properly used in CCCS-CIC-AndMal2022. We take 12 families (except labels of *Zero-day* and *No Category*) for the experiment, then split the data in the proportion of 7:3 for training and testing. The details are given in Table 1.

Table 1
The detail of dataset.

Category	Number of samples	Number of samples in training set	Number of samples in testing set
Adware	47,210	32,969	14,241
Backdoor	1,538	1,040	498
FileInfector	669	482	187
PUA	2,051	1,431	620
Ransomware	6,202	4,379	1,823
Riskware	97,349	68,242	29,107
Scareware	1,556	1,100	456
Trojan	13,559	9,537	4,022
Banker	887	606	281
Dropper	2,302	1,599	703
SMS	3,125	2,183	942
Spy	3,540	2,423	1,117
sum	**179,988**	**125,990**	**53,998**

Parameters: The adaptive procedure of the proposed method is as follows. We first set the parameters of CTGAN as batch size = 50, epochs = 100. According to the Eq. (8), we determine the baseline amount of 1469. During the training procedure, it is found that SVM's discrimination rate is over 5%. Therefore, the parameters of CTGAN are tuned as batch size = 70, 60, 40, 30, 20, 10, and the epochs are also tuned from 120 to 20. Finally, when batch size = 20, epochs = 50, the lowest discrimination rate coming to 0.47 (under 5%). Thus, it is considered that the generated data are qualified and the parameters are fixed. A part of the procedure is shown in Table 6 and 7.

Comparison experiment: To evaluate the effectiveness of the proposed method, we have designed two kinds of experiments: longitudinal comparative experiment and horizontal comparative experiment.

Longitudinal experiments change the structure of the method to verify the validity of the proposed structure. In detail, we retain steps 1 and 2 of SynDroid and just test different fundamental classifiers to find the best one. Then we remove the generator of SynDroid and instead use a parameter named class weight, which is commonly applied in the Scikit-learn package when the dataset is not balanced. It is defined as

$$class\ weight = \frac{n\_samples}{n\_classes \times i\_samples} \tag{11}$$

where *n_samples* is the amount of all samples, *n_classes* is the kinds of classes in the dataset and *i_samples* is the number of samples in certain class *i*. Cost-sensitive learning will calculate every class weight to construct a cost matrix. It tunes the training weight of every class according to the matrix. Oppositely, we just tune certain classes' chosen by the KS-CIR test (not every) training weight to evaluate the effectiveness of the KS-CIR test. After that, we remove steps 1 and 2 of SynDroid, and instead apply SMOTE, ADASYN, BorderlineSMOTE and KmeansSMOTE comparing with the GAN model to enhance the dataset, taking the best classifier found in the former step to make the classification.

Table 2
K-S CIR test.

Category	K-S score	CIR	Number of samples
Adware	**0.9011**	2.07	32,969
Backdoor	0.8523	**65.62**	1,040
FileInfector	**0.8936**	**141.58**	482
PUA	**0.9099**	**47.69**	1,431
Ransomware	**0.9052**	15.58	4,379
Riskware	**0.8971**	1	68,242
Scareware	0.8849	**62.04**	1,100
Trojan	0.8826	7.16	9,537
Banker	**0.8970**	**112.61**	606
Dropper	0.8830	42.68	1,599
SMS	**0.9070**	31.26	2,183
Spy	**0.9020**	28.16	2,423
average	*0.8930*	*46.45*	-

Additionally, to verify whether SVM can replace manual checking to help the self-adaptive process of the GAN model, groups of data with different batch sizes and epochs are generated. SVM will discriminate every group of generated data and get a rate. Then we randomly select two features of real and generated data and calculate the statistical properties. Our purpose is to see the relationship between SVM's rate and manual-calculated statistical results: whether the lower discrimination rate represents the generated data's higher similarity.

Horizontal experiments compare the proposed method with other complicated models, including XGBoost (Chen and Guestrin, 2016), BiLSTM with attention and DiDroid (Rahali et al., 2020). XGboost is a strong classifier using gradient-boosted decision trees algorithm, it is widely used in machine learning competitions. BiLSTM is a recurrent neural network used to process sequence data. It considers both the front input and the back input (LSTM only passed from left to right). BiLSTM trains with an attention mechanism used to emphasize features, the results will be more accurate and are extensively used in classification tasks (Shen et al., 2022). DiDroid is the method proposed associating with CCCS-CIC-AndMal-2020, so it is proper to compare SynDroid with it.

Evaluation metrics: The evaluation metrics are accuracy, precision, recall, and F1 score. They are defined as followed:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{12}$$

$$Precision = \frac{TP}{TP + FP} \tag{13}$$

$$Precision = \frac{TP}{TP + FN} \tag{14}$$

$$F1 = \frac{2 \times Precission \times Recall}{Precission + Recall} \tag{15}$$

where TP, TN, FP and FN are all defined in multi-classification tasks, meaning one class *vs* others while calculating.

4.2. K-S CIR test

We apply the K-S CIR test on 12 Android malware classes and the result is shown in Table 2. According to Eq. (6) and Eq. (7), we get two lists of categories whose K-S score and CIR are above the average. List 1 of the K-S score includes *Adware*, *File Infector*, *PUA*, *Ransomware*, *Riskware*, *Banker*, *SMS* and *Spy*. List 2 of CIR includes *Backdoor*, *File Infector*, *PUA*, *Scareware*, and *Banker*. The intersection of the two lists is *File Infector*, *PUA* and *Banker*, the baseline amount calculated according to Eq. (8) is 1469. After generating the *File Infector*, *PUA* and *Banker* samples, the average of CIR decreased from 46.65 to 32.91.

It can be seen that the KS-CIR test does not merely choose classes with small quantities. For example, *Backdoor* and *Scareware* both are with CIR over 60, but according to the K-S score they have prominent features, meaning they can be easily distinguished from other classes although they are in the minority.

Table 3
Comparison of different classifiers after using SynDroid.

Model	Accuracy	Recall	Precision	F1-score
K-Nearest Neighbors	0.8886	0.7473	0.7357	0.7359
MLP	0.9179	0.7446	0.8154	0.7697
Decision Tree	0.9217	0.7459	0.8488	0.7857
Random Forest	0.9261	0.7391	0.8864	0.7864
SynDroid-KNN	0.9100	0.7514	0.7672	0.7551
SynDroid-MLP	0.9294	0.7910	0.8207	0.8020
SynDroid-DT	0.9310	0.8098	0.8176	0.8124
SynDroid-RF	**0.9431**	**0.8219**	**0.8866**	**0.8488**

Table 4
Class weight result with Eq. (11).

Category	class weight
Adware	0.32
Backdoor	10.10
FileInfector	**21.78**
PUA	**7.34**
Ransomware	2.39
Riskware	0.15
Scareware	9.54
Trojan	1.10
Banker	**17.33**
Dropper	6.57
SMS	4.81
Spy	4.33

Table 5
Comparison of Cost-sensitive Learning and KS-CIR test.

Method	Accuracy	Recall	Precision	F1-score
Cost-sensitive	93.57	**84.65**	82.56	82.87
KS-CIR test	**94.25**	82.56	**87.61**	**84.51**

4.3. Performance comparison

4.3.1. Longitudinal experiments

The results of longitudinal experiments are as followed. In Table 3, we test different bottom classifiers including K-Nearest-Neighbors (KNN), Multilayer Perceptron (MLP), Decision Tree (DT), and Random Forest(RF). It can be seen that every classifier gets better performance after using SynDroid to enhance the minority data. The SynDroid with RF gets the best performance at all metrics, so the classifier is determined.

Then, we calculate every class weight with Eq. (11) and the result is shown in Table 4. Based on RF, we use cost-sensitive learning to tune the training weight of every class, comparing it with using the KS-CIR test just to tune the training weight of *File Infector*, *PUA* and *Banker*. The result is shown in Table 5. We can see that cost-sensitive learning gets the best recall, but KS-CIR test wins cost-sensitive learning in accuracy, precision and F1-score. This result shows that our KS-CIR test can effectively choose the classes which are most in need to be enhanced not just based on the samples' amount.

After that, the KS-CIR test and CTGAN are removed. Instead, we use traditional oversampling methods to enhance the dataset. The result in Table 8 shows that SMOTE and BorderlineSMOTE with RF even perform worse than SynDroid-DT in all metrics. ADASYN-XGB gets the best precision but loses accuracy, precision, and F1-score to KmeansSMOTE and SynDroid. KmeansSMOTE is the best traditional over-sampling method above all, but SynDroid wins it by about 1% in accuracy, 0.5% in recall, 5% in precision, and 3% in F1-score.

Additionally, we generate 5 groups of data of *FileInfector* and apply SVM to distinguish them. Firstly, we choose the epoch as 50 and tune different batch sizes, the results are shown in Table 6. The lowest discrimination rate comes to the batch size of 20. Then, we choose

Table 6
Generated data with epoch 50.

batch size	epoch	SVM's discrimination rate
10	50	0.109
20	50	**0.047**
30	50	0.076

Table 7
Generated data with batch size 20.

batch size	epoch	SVM's discrimination rate
20	30	0.181
20	50	**0.047**
20	70	0.083

Table 8
Comparison of oversampling methods and SynDroid.

Model	Accuracy	Recall	Precision	F1-score
SMOTE with RF	0.9157	0.7838	0.7629	0.7545
SMOTE with XGB	0.8772	0.7729	0.6862	0.7159
SMOTE with MLP	0.9058	0.7797	0.7151	0.7309
BorderlineSMOTE with RF	0.9157	0.7825	0.7632	0.7542
BorderlineSMOTE with XGB	0.8905	0.7566	0.6859	0.7023
BorderlineSMOTE with MLP	0.9042	0.7751	0.7100	0.7259
ADASYN with RF	0.9207	0.8350	0.7819	0.8012
ADASYN with XGB	0.8903	0.8139	0.6969	0.7425
ADASYN with MLP	0.9121	**0.8384**	0.7394	0.7793
KmeansSMOTE with RF	0.9339	0.8165	0.8367	0.8191
KmeansSMOTE with XGB	0.9158	0.8021	0.7801	0.7872
KmeansSMOTE with MLP	0.9329	0.8279	0.8305	0.8278
SynDroid	**0.9431**	0.8219	**0.8866**	**0.8488**

Table 9
Horizontal experiment result.

Model	Accuracy	Recall	Precision	F1-score
BiLSTM with Attention	0.9278	0.7921	0.8367	0.8118
XGBoost	0.9394	0.8225	0.8804	0.8473
DiDroid	0.8213	**0.8487**	0.8150	0.8297
SynDroid	**0.9431**	0.8236	**0.8865**	**0.8495**

the batch size of 20 and tune different epochs, the results are shown in Table 7. Generated data with a batch size of 20 and epoch of 50 also get the lowest discrimination rate. After that, we randomly select two features of the data to calculate statistical characteristics and make boxplots shown in Fig. 5. It is clear that data distribution with a batch size of 20 and epoch of 50 is most similar to the real data compared with others.

4.3.2. Horizontal experiment

In the horizontal experiment, we compare our method with XGBoost, BiLSTM with Attention mechanism, and DiDroid. The normalized confusion matrix is shown in Figs. 6, 7, 8, 9, and the final result is shown in Table 9. We can conclude from Table 9 that DiDroid gets the best recall and SynDroid gets the best accuracy, precision, and F1-score. Also, tree classifiers (XGBoost and SynDroid) perform better than neural network classifiers (BiLSTM and DiDroid). In the figures, it can be seen that minority classes (Riskware and Adware) always lead to the best classification result (over 90%). But superiority in quantity does not lead to better results of necessity, all four methods perform better on Ransomware (4379 samples) than Trojan (9537 samples), that's the reason why we should focus on both quality and quantity to balance the dataset (see Figs. 6, 7, 8, 9).

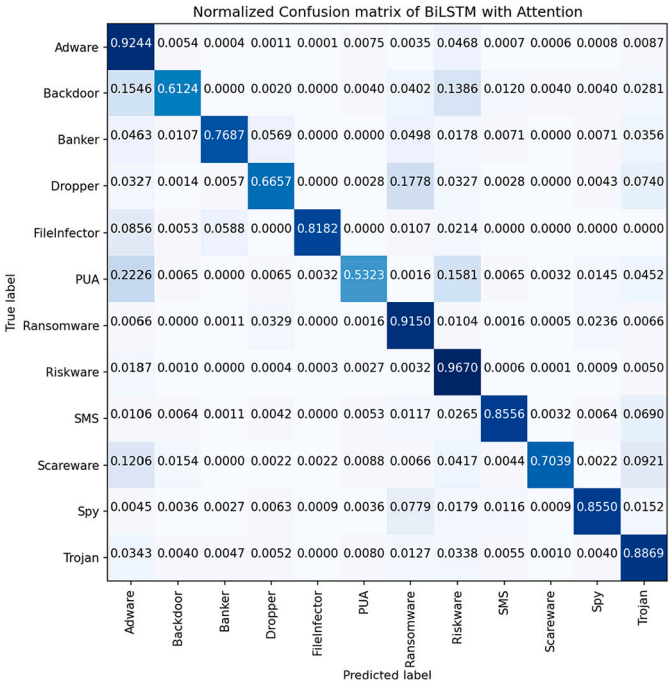


Fig. 6. BiLSTM.

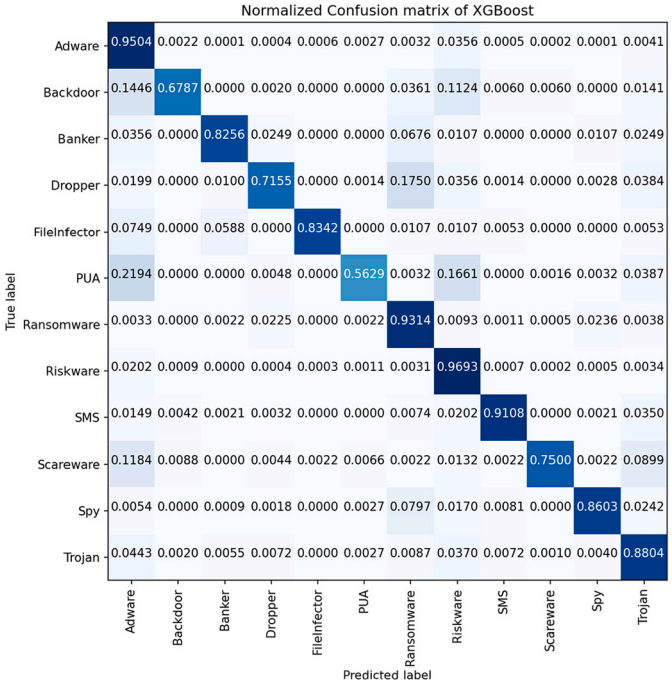


Fig. 7. XGBoost.

4.4. Discussion

In the longitudinal experiment, Table 3 reveals that after using SynDroid, all models perform better than before. The effectiveness of SynDroid is verified. The results in Table 5 show that cost-sensitive gets the best recall, SynDroid gets the best accuracy, precision, and F1-score. The reason is that cost-learning pays the most attention to the minority, and lots of samples are misclassified as the minority class. Correspondingly, they will lose a lot in accuracy, precision, and F1 score. The fact that the KS-CIR test wins cost-sensitive learning in accuracy, precision and F1-score demonstrates KS-CIR can truly determine certain classes

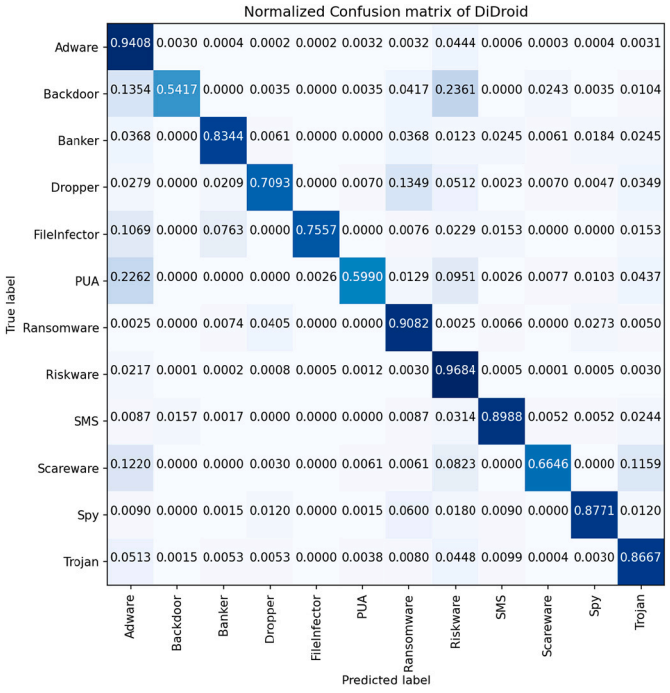


Fig. 8. Didroid.

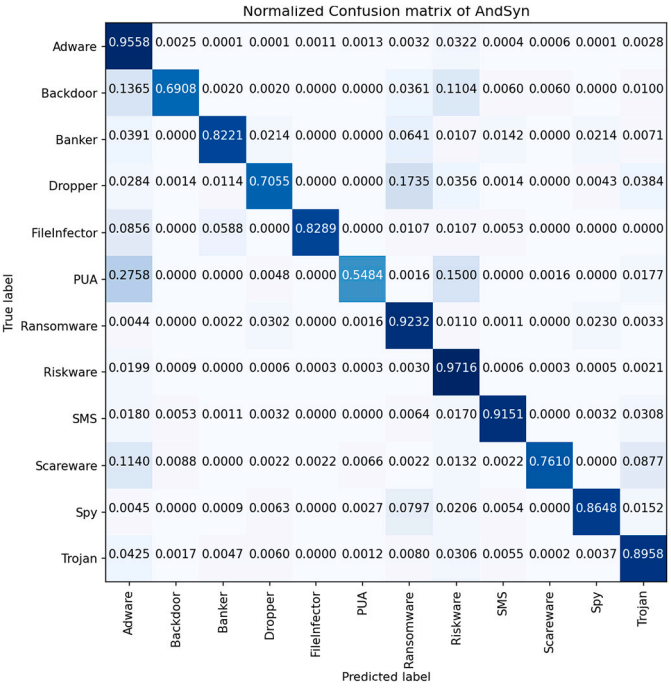


Fig. 9. SynDroid.

needed the most attention not only by quantity gap. Therefore, tuning the training weight of these classes is better than that of every class. The result also reveals that just tuning training weight is not enough while processing a dataset with great CIR, and data-level technology should be focused on.

Comparing Table 3 with Table 8 we can find that SMOTE and BorderlineSMOTE with RF perform worse than SynDroid with DT in all metrics, even though RF is a more advanced algorithm. One reason is that our samples have 230 features (after being selected) and these traditional methods cannot generate qualified samples in such high dimensions. Another reason is that these methods simply focus on the amount gap between minority and majority, thus they have to generate samples from the amount of 100-level to 10K-level, which is extremely hard. ADASYN-RF gets better precision than SynDroid, but loses a lot in Recall and F1-score. KmeansSMOTE-RF performs better than other oversampling methods, because it is a newer algorithm and we find that it merely duplicates the same samples to balance the dataset. Thus it only changes the number of samples but the model does not learn more information about the minority at all.

The results in Tables 6–7 show that the lowest SVM discrimination comes to generated data with a batch size of 20 and epochs of 50. And we can see clearly that in Figs. 5, the data with a batch size of 20 and epochs of 50 have the closest statistical characteristics to real data. Therefore, it can be concluded that SVM can really replace manual checking to help the self-adaptive procession of the GAN model: the less rate usually means generated data's better quality. We can properly tune the GAN model with the guide of SVM's discrimination rate, rather than calculating every group of data's statistical characteristics to measure whether the generated data is good or not. The use of SVM outside GAN consumes less time and is effective.

In the horizontal experiment, we can see all these methods get satisfying results but SynDroid performs better. Classifiers based on Tree algorithms (XGBoost and SynDroid) get a better result than those based on neural networks (Didroid and BiLSTM with Attention mechanism) because the Android malware classification task essentially is a yes-or-no problem, which is very suitable for Tree model with a branching structure. Neural networks like CNN and BiLSTM are all under the control of loss function, which will change the yes-or-no problem to the complicated cost problem and even gets a worse result. We can also find that all these methods can classify the majority (*Adware*, *Riskware*) with an accuracy of over 95%, better than those with fewer quantities. However, the superiority in quantity does not necessarily lead to a better result, for instance, the classifier performs better on *FileInfector* than *PUA* and *Dropper*.

5. Conclusion and future work

In this paper, we propose SynDroid, an Android malware classification method using GAN to augment the dataset. To determine which classes in the dataset needed to be enhanced, we propose the KS-CIR test, combining the K-S test and Class Imbalance Ratio, taking both samples' quality and quantity into consideration. With the KS-CIR test and CTGAN, we augment our dataset and get good performance at CCCS-CIC-AndMal2020.

However, SynDroid still exists limitations. SynDroid alleviates the imbalance problem of a dataset to some extent, but it cannot thoroughly solve the problem. It uses GAN to enhance the data for a better classification result but consumes more time. The problems of the GAN model itself are not solved despite the use of SVM has discarded the bad results. In the future, we will improve the method and experiment on more datasets so that it can be widely used in Android malware detection. Also, K-S test can only measure certain classes of data. We still lack a metric to measure the quality of the whole data set. In the future, we want to propose a method combining the improvement of classification results and statistical property of the data set (ratio of errors, number of empty values, amounts of dark data, information volume and so on). With the guide of this method, the data enhancement of malware can be more scientific and mathematical.

CRediT authorship contribution statement

Junhao Li: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Writing – original draft. **Junjiang He:** Conceptualization, Project administration, Supervision, Writing – review & editing. **Wenshan Li:** Formal analysis, Methodology, Supervision, Validation. **Wenbo Fang:** Data curation, Formal analysis. **Geying Yang:**

Data curation, Formal analysis. **Tao Li:** Funding acquisition, Supervision.

Declaration of competing interest

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

Data availability

The data that has been used is confidential.

Acknowledgement

This work was supported in part by the National Key Research and Development Program of China (No. 2020YFB1805400); in part by the National Natural Science Foundation of China (No. U19A2068, No. 62032002, and No. 62101358); in part by the Youth Natural Science Foundation of Sichuan (No. 2023NSFSC1395); in part by the Sichuan Province science and technology plan key research and development project (No. 2023YFG0294); in part by the Sichuan University Post doctoral Science Foundation (No. 2023SCU12127); Fundamental Research Funds for the Central Universities (Grant No. SCU2021D052).

References

Smartphone market share in 2022. https://gs.statcounter.com/os-market-share/mobile/worldwide.

Goldoson issue. https://www.bleepingcomputer.com/news/security/android-malware-infiltrates-60-google-play-apps-with-100m-installs/.

Android malwares evolution in 2021. https://securelist.com/mobile-malware-evolution-2021/105876/.

Data augmentation. https://en.wikipedia.org/wiki/Data_augmentation.

Kolmogorov-Smirnov test. https://en.wikipedia.org/wiki/Kolmogorov-Smirnov_test.

Aafer, Y., Du, W., Yin, H., 2013. Droidapiminer: mining api-level features for robust malware detection in Android. In: International Conference on Security and Privacy in Communication Systems.

Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., Rieck, K., 2014. Drebin: effective and explainable detection of Android malware in your pocket. In: Network & Distributed System Security Symposium.

Canfora, G., Medvet, E., Mercaldo, F., Visaggio, C.A., 2015. Detecting Android malware using sequences of system calls. In: Workshop on ACM Press the 3rd International, pp. 13–20.

Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P., 2002. Smote: synthetic minority over-sampling technique. vol. 1. https://doi.org/10.1613/JAIR.953.

Chen, B.C., Ren, Z.R., Yu, C., Hussain, I., Liu, J.T., 2019. Adversarial examples for cnn-based malware detectors. IEEE Access 7, 54360–54371. https://doi.org/10.1109/ACCESS.2019.2913439.

Chen, H., 2021. Challenges and corresponding solutions of generative adversarial networks (gans): a survey study. J. Phys. Conf. Ser. 1827, 012066.

Chen, J., Alalfi, M.H., Dean, T.R., Zou, Y., 2015. Detecting Android malware using clonrticlee detection. https://doi.org/10.1007/s11390-015-1573-7.

Chen, L., Hou, S., Ye, Y., Xu Droideye, S., 2018. Fortifying security of learning-based classifier against adversarial Android malware attacks. In: 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), pp. 782–789.

Chen, T., Guestrin, C., 2016. Xgboost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'16. Association for Computing Machinery, pp. 785–794.

Chen, Y.M., Yang, C.H., Chen, G.C., 2021. Using generative adversarial networks for data augmentation in Android malware detection. In: 2021 IEEE Conference on Dependable and Secure Computing (DSC).

Feng, P., Ma, J., Cong, S., Xu, X., Ma, Y., 2018. A novel dynamic Android malware detection system with ensemble learning. IEEE Access 6, 30996–31011. https://doi.org/10.1109/ACCESS.2018.2844349.

Feng, R., Chen, S., Xie, X., Ma, L., Meng, G., Liu, Y., Lin, S.W., 2019. Mobidroid: a performance-sensitive malware detection system on mobile platform. In: 2019 24th International Conference on Engineering of Complex Computer Systems (ICECCS).

Feng, R., Jing, Q.L., Chen, S., Lin, S.W., Liu, Y., 2020. Seqmobile: a sequence based efficient Android malware detection system using rnn on mobile devices. arXiv: 2011.05218.

Feng, R., Chen, S., Xie, X., Meng, G., Lin, S.-W., Liu, Y., 2021. A performance-sensitive malware detection system using deep learning on mobile devices. 16, 1563–1578. https://doi.org/10.1109/TIFS.2020.3025436.

Fereidooni, H., Conti, M., Yao, D., Sperduti, A., 2016. Anastasia: Android malware detection using static analysis of applications. In: Ifip International Conference on New Technologies, pp. 1–5.

Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial nets. In: Proceedings of the 27th International Conference on Neural Information Processing Systems, vol. 2, NIPS'14. MIT Press, pp. 2672–2680.

He, H., Yang, B., Garcia, E.A., Li, S., 2008. Adasyn: adaptive synthetic sampling approach for imbalanced learning. In: Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on.

Ho, T.K., 1995. Random decision forests. In: Proceedings of 3rd International Conference on Document Analysis and Recognition, vol. 1, pp. 278–282.

Hu, W., Tan, Y., 2017. Generating adversarial malware examples for black-box attacks based on gan. https://doi.org/10.48550/arXiv.1702.05983.

Hui, H., Wang, W.Y., Mao Borderline-smote, B.H., 2005. A new over-sampling method in imbalanced data sets learning. In: Proceedings of the 2005 International Conference on Advances in Intelligent Computing - Volume Part I.

Jian, Y., Kuang, H., Ren, C., Ma, Z., Wang, H., 2021. A novel framework for image-based malware detection with a deep neural network. 109, 102400. https://doi.org/10.1016/j.cose.2021.102400.

Keyes, D.S., Li, B., Kaur, G., Lashkari, A.H., Gagnon, F., Massicotte, F., 2021. Entroplyzer: Android malware classification and characterization using entropy analysis of dynamic characteristics. In: 2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS), pp. 1–12.

Kim, J.Y., Bu, S.J., Cho, S.B., 2018. Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders. https://doi.org/10.1016/j.ins.2018.04.092. pp. 83–102.

Last, F., Douzas, G., Bacao, F., 2017. Oversampling for imbalanced learning based on k-means and smote. https://doi.org/10.48550/arXiv.1711.00837.

Lei, X., Veeramachaneni, K., 2018. Synthesizing tabular data using generative adversarial networks. https://doi.org/10.48550/arXiv.1811.11264.

Liu, P., Wang, W., Luo, X., Wang, H., Liu, C., 2021. Nsdroid: efficient multi-classification of Android malware using neighborhood signature in local function call graphs. vol. 20. https://doi.org/10.1007/s10207-020-00489-5.

Qiao, M., Sung, A.H., Liu, Q., 2016. Merging permission and api features for Android malware detection. In: 2016 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI), pp. 566–571.

Rahali, A., Lashkari, A.H., Kaur, G., Taheri, L., Gagnon, F., Massicotte, F., 2020. Didroid: Android malware classification and characterization using deep image learning. In: ICCNS 2020. Association for Computing Machinery, New York, NY, USA, pp. 70–82.

Renjith, G., Laudanna, Sonia, Aji, S., Corrado, Aaron Visaggio, Vinod, P., 2021. Gangmam: gan based engine for modifying Android malware. https://doi.org/10.48550/arXiv.2109.13297.

Rosenberg, I., Shabtai, A., Elovici, Y., Rokach, L., 2021. Adversarial Machine Learning Attacks and Defense Methods in the Cyber Security Domain, vol. 54. Association for Computing Machinery.

Shen, G., Chen, Z., Wang, H., Chen, H., Wang, S., 2022. Feature fusion-based malicious code detection with dual attention mechanism and bilstm. 119, 102761. https://doi.org/10.1016/j.cose.2022.102761.

Suarez-Tangil, G., Dash, S.K., Ahmadi, M., Kinder, J., Cavallaro, L., 2017. Droidsieve: fast and accurate classification of obfuscated Android malware. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, pp. 309–320.

Vasan, D., Alazab, M., Wassan, S., Naeem, H., Zheng, Q., 2020. Imcfn: image-based malware classification using fine-tuned convolutional neural network architecture. 171, 107138. https://doi.org/10.1016/j.comnet.2020.107138.

Wang, L.C.W., lv, Shaohua, Liu, Jiqiang, Chang, Xiaolin, Wang, Jinqiang, 2020. On the combination of data augmentation method and gated convolution model for building effective and robust intrusion detection. vol. 3. https://doi.org/10.1186/s42400-020-00063-5.

Wu, B., Chen, S., Gao, C., Fan, L., Lyu, M.R., 2021. Why an Android app is classified as malware: toward malware classification interpretation. 30, 1–29. https://doi.org/10.1145/3423096.

Wu, D.-J., Mao, C.-H., Wei, T.-E., Lee, H.-M., Wu, K.-P., 2012. Droidmat: Android malware detection through manifest and api calls tracing. In: 2012 Seventh Asia Joint Conference on Information Security, pp. 62–69.

Xiao, X., Zhang, S., Mercaldo, F., Hu, G., Sangaiah, A.K., 2018. Android malware detection based on system call sequences and lstm. https://doi.org/10.1007/s11042-017-5104-0.

Xie, N., Zeng, F., Qin, X., Yu, Z., Lv, C., 2018. Repassdroid: automatic detection of Android malware based on essential permissions and semantic features of sensitive apis. In: 2018 International Symposium on Theoretical Aspects of Software Engineering (TASE).

Zhou, Z.H., Liu, X.Y., 2006. Training cost-sensitive neural networks with methods addressing the class imbalance problem. 18, 63–77. https://doi.org/10.1109/TKDE.2006.17.

Junhao Li received B.Eng from Civil Aviation University of China, Tianjing, China, in 2021. He is currently pursing the M.A degree with the School of Cyber Science and Engineering, Sichuan University, Chengdu, China. His current research interests include web security and machine learning.

Junjiang He received Ph.D. degree in Sichuan University, Chengdu, Sichuan, China, in 2021. He is currently an Assistant Research Fellow in the School of Cyber Science and Engineering, Sichuan University, Chengdu, China. His current research interests include artificial immune system, cyber-physical system security, information security, cloud storage and data mining.

Wenshan Li received her M.S. degree from University of California, Los Angeles, in 2019. She is currently a Lecturer in the School of Cyber Science and Engineering, Chengdu University of Information Technology, Sichuan, China. Her current research interests include data science, machine learning and bioinformatics.

Wenbo Fang received his MS degree from the School of Software Engineering at Xinjiang University Urumqi, Xinjiang, China, in 2020. He is recently pursuing a PhD at the School of Cyberspace Security at Sichuan University, Chengdu, Sichuan, China. His current research interests include artificial immunity, information security, mobile security and data security.

Geying Yang received her MS degree from the School of Computer and Information Engineering at Henan University, Kaifeng, Henan, China, in 2020. She is currently pursuing the PhD degree with the National Institute of Cybersecurity, Wuhan University, Wuhan, China. Her current research interests include artificial immune system, evolutionary computation, multi-objective optimization and its real-world applications.

Tao Li received his Ph.D. degree in computer science from the University of Electronic Science and Technology of China, in 1994. He is currently a Professor in the School of Cyber Science and Engineering, Sichuan University, China. He is the Chief Scientist of the National Key Research and Development Plan for Cyberspace Security. He is also an editorial board member of Immune Computation and several other international academic journals. His main research interests include network security, artificial immune systems, cloud computing, and cloud storage. He has published nearly 300 papers in IEEE, ACM, Chinese Science, Science Bulletin, Natural Science Progress and other important journals and academic conferences.