IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received June 4, 2020, accepted June 18, 2020, date of publication July 1, 2020, date of current version July 20, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3006143

A Review of Android Malware Detection Approaches Based on Machine Learning

KAIJUN LIU[1,2], SHENGWEI XU[3], GUOAI XU[1,2], MIAO ZHANG[1,2], DAWEI SUN[4], AND HAIFENG LIU[5]

[1]School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China
[2]National Engineering Laboratory of Mobile Network Security, Beijing University of Posts and Telecommunications, Beijing 100876, China
[3]Information Security Research Institute, Beijing Electronic Science and Technology Institute, Beijing 100070, China
[4]Research Center for Intelligent Software Security, Beijing Softsec Technologies Company Ltd., Beijing 100876, China
[5]Beijing Information Security Test and Evaluation Center, Beijing 100101, China

Corresponding author: Guoai Xu (xga@bupt.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB0803605, in part by the Basic Scientific Research Projects of National Defense Science, Technology, and Industry Technology under Grant JSZL2017601C-1, and in part by the National Natural Science Foundation of China under Grant 61897069 and Grant 61831003.

ABSTRACT Android applications are developing rapidly across the mobile ecosystem, but Android malware is also emerging in an endless stream. Many researchers have studied the problem of Android malware detection and have put forward theories and methods from different perspectives. Existing research suggests that machine learning is an effective and promising way to detect Android malware. Notwithstanding, there exist reviews that have surveyed different issues related to Android malware detection based on machine learning. We believe our work complements the previous reviews by surveying a wider range of aspects of the topic. This paper presents a comprehensive survey of Android malware detection approaches based on machine learning. We briefly introduce some background on Android applications, including the Android system architecture, security mechanisms, and classification of Android malware. Then, taking machine learning as the focus, we analyze and summarize the research status from key perspectives such as sample acquisition, data preprocessing, feature selection, machine learning models, algorithms, and the evaluation of detection effectiveness. Finally, we assess the future prospects for research into Android malware detection based on machine learning. This review will help academics gain a full picture of Android malware detection based on machine learning. It could then serve as a basis for subsequent researchers to start new work and help to guide research in the field more generally.

INDEX TERMS Android security, malware detection, machine learning, feature extraction, classifier evaluation.

I. INTRODUCTION

Since Android was released in 2008, it has become the most popular operating system for smart mobile devices. In 2019, about 86.6% of smartphones sold globally were based on Android [1]. By the end of April 2020, there were more than 2.8 million applications on Google Play, which is the official store for Android applications [2]. Due to various factors, such as the open ecological mode of Android applications, its coarse-grained permission management, and the ability to invoke third-party code, many security attack surfaces are present, which seriously threatens the integrity of Android applications. Statistics show that in 2016 alone, more than 3.25 million Android apps that were infected with malware were discovered, which means that a new Android malware app was found roughly every 10 seconds [3]. To ensure the security of the Android ecosystem, a variety of solutions have been proposed, including application reinforcement, vulnerability detection, developer reviews, and malware detection [4]. Among the various security options, Android malware detection is a widely used security protection method that can prevent malware from being released into the Android application marketplace or being installed and used. Based on previous research, Android malware detection technology can be divided into three categories: static detection, dynamic detection, and hybrid detection [5]–[7]. Static detection is based on the analysis of suspect code without running the Android application. It can

The associate editor coordinating the review of this manuscript and approving it for publication was Ioannis Schizas.

achieve high code coverage but faces many countermeasures such as code obfuscation and dynamic code loading. Conversely, dynamic detection involves the analysis of the Android application by running the code. This can expose risks that are not easy to discover by static analysis, but the computational resources and time cost of dynamic detection are relatively high. Hybrid detection is a method that combines static detection and dynamic detection to achieve a balance between detection effectiveness and efficiency. Machine learning theory is widely applied in the detection of Android malware, whether based on static, dynamic, or hybrid analysis approaches. Compared with traditional methods, such as signature-based malware detection, which is based on identifying specific patterns of known malware, machine learning-based detection has the ability to detect previously unseen types of malware [8] and can provide better performance in detection efficacy and efficiency [9], [10]. Some previous studies have discussed Android malware detection approaches based on machine learning. However, there are some limitations in the surveyed research, including the now outdated literature on which previous reviews were based, the narrow scope of studies, and the lack of discussion regarding some controversial content. To overcome these limitations, this paper presents a systematic overview of research within this specific area of work. The main contributions of this paper are summarized as follows.

(1) We present a systematic and categorized overview of machine learning approaches to Android malware detection. The paper briefly covers some of the wider background of Android applications but focuses on key aspects of machine learning such as sample acquisition, data preprocessing, feature selection, machine learning model theory, and evaluation of detection effectiveness.

(2) We fill some research gaps in previous reviews on machine learning methods for Android malware detection. These are important aspects in the field of machine learning but are rarely mentioned in previous reviews. Examples include data preprocessing and reliability estimation.

(3) We point out some areas of disagreement or neglect in the field of Android malware detection based on machine learning, and present our own assessments based on our review of the relationship between data preprocessing and feature selection, the classification of machine learning algorithms, and the relationship between neural networks and deep learning.

(4) We further explicate the limitations of machine learning approaches in Android malware detection and provide insights for potential new research directions.

As shown in Table 1, this review differs from several previous works in this discipline. It is not a general study of Android malware detection [15], [23] or Android security [35], [37] but instead systematically focuses on key perspectives on machine learning approaches used in Android malware detection. In some papers, although machine learning methods are included, the information is scattered throughout the article to support different objectives. For example, researchers in Ref. [24] give an overview of malware detection using data mining techniques. This article is from the perspective of data mining, and although some mainstream machine learning methods are analyzed, it does not pay attention to the key aspects of the whole machine learning process. Furthermore, some similar reviews only focus on one or two aspects of machine learning approaches in Android malware detection. For example, researchers in Ref. [30] primarily summarize feature selection in mobile malware detection, while researchers in Ref. [12] mainly summarize various machine learning methods.

From the above analysis, there is a clear need to conduct a review that gives a more general and comprehensive understanding of the state-of-the-art research in this field, with the end goal being to help motivate and direct future research. This review may thus be useful to a wide range of readers. The rest of this paper is organized as follows. Section II describes our method of literature collection. Section III outlines some background on Android applications, including the Android system architecture, security mechanisms, and classification of Android malware. Section IV presents a comprehensive survey of Android malware detection approaches based on machine learning from key perspectives such as sample acquisition, data preprocessing, feature selection, machine learning models, algorithms, and evaluation of detection effectiveness. Section V suggests some research directions and challenges for future work. Finally, Section VI gives our conclusions.

II. METHOD OF LITERATURE COLLECTION

For a literature review, it is very important to collect all the relevant literature. We consider it necessary in this paper to briefly describe our method of literature collection. Our process of literature collection was as follows.

(1) Based on the theme of this review, we determined the most relevant information we wanted to collect. Obviously, the information that needs to be collected is focused on machine learning approaches for Android malware detection.

(2) We determined the search keywords in terms of the information to be collected. There are three formats of keywords. First, because the topic of this review involves both machine learning and Android malware detection, we summarize some keyword combinations such as ''machine learning + Android malware detection'' and ''machine learning + Android malware''. Second, we believe that machine learning is a more extensive field compared with the detection of Android malware, and therefore we expanded the scope to directly search keywords such as ''Android malware detection''. Finally, we tried to find information about machine learning from previous reviews of Android security, searching for keywords such as ''review/survey/overview + Android security'' and ''review/survey/overview + Android malware''.

(3) We selected the data source to search using the keywords. Most of the reviewed literature was from top

TABLE 1. Comparison of recent reviews having overlapping coverage with this article. (√ = Having content, * = Little to no content, × = No content).

Content of Review	Year	Reference	Sample Acquisition	Data Preprocessing	Feature Selection	Feature Type	Machine Learning Method	Evaluation of Detection Effectiveness		
								Division of Dataset	Evaluation of Classifier Performance	Reliability Estimation of Evaluation Results
Android Malware Detection based on Machine Learning	2020	This Article	√	√	√	√	√	√	√	√
	2019	[6]	×	×	*	√	√	×	√	×
	2019	[11]	×	×	*	√	√	×	√	×
	2019	[12]	×	×	*	√	√	×	√	×
	2018	[13]	√	*	×	√	√	×	√	×
	2016	[14]	√	×	×	√	√	√	√	×
Android Malware Detection	2020	[15]	*	×	×	*	*	×	√	×
	2019	[16]	×	×	×	*	*	×	√	×
	2019	[17]	√	×	×	√	√	*	√	×
	2019	[18]	×	×	×	*	√	×	√	×
	2018	[19]	×	*	*	√	√	×	√	×
	2018	[20]	√	×	×	*	√	×	√	×
	2018	[21]	×	×	×	*	×	×	√	×
	2018	[22]	×	×	×	√	×	*	√	×
	2018	[9]	√	×	*	*	√	×	√	×
	2017	[23]	√	×	√	√	*	×	*	×
	2017	[24]	*	×	√	√	√	*	√	×
	2017	[25]	×	×	√	√	√	×	*	×
	2017	[26]	×	×	×	√	*	×	*	×
	2016	[27]	×	×	×	√	√	*	*	×
	2016	[28]	×	×	×	√	√	*	√	×
	2015	[29]	√	×	×	√	*	×	×	×
	2015	[30]	√	×	√	√	√	×	√	×
	2015	[31]	×	×	×	√	√	×	×	×
	2015	[32]	×	×	√	√	√	×	*	×
	2014	[33]	×	×	×	√	√	×	√	×
	2014	[34]	×	×	×	*	*	×	*	×
Android (Mobile) Security	2019	[35]	√	×	×	√	√	×	*	×
	2017	[8]	×	×	×	√	√	×	×	×
	2017	[36]	×	×	×	√	×	×	×	×
	2016	[37]	√	×	×	√	√	×	*	×
	2015	[4]	×	×	×	√	√	×	√	×
	2014	[38]	√	×	*	√	√	×	*	×
	2012	[39]	×	×	*	√	√	×	*	×

conferences or mainstream journals, suggesting that our study has considered the important relevant works. The following online repositories were searched: ACM Digital Libraries [40], Science Direct [41], Web of Science [42], IEEE Xplore Digital Library [43], Cornell University Library [44], and SpringerLink [45]. In addition to the

direct search of these online repositories, we also searched some widely used third-party channels, including Google Scholar [46], ResearchGate [47], and Academia [48].

(4) We determined the published date range of the collected literature and conducted a preliminary search. Android was released in 2008, and its security has attracted more and more attention in recent years; hence, we focused on research papers dated no more than 10 years ago, especially those published in the last five years. Additionally, in order to ensure that the literature analysis and summary were relatively fixed, we set the deadline of literature collection as April 30, 2020, which is in accord with the actual time period within which this article was written.

(5) We excluded unsuitable literature from the preliminary literature collection. The title of some of the collected literature was fascinating, but in fact its content was unfortunately inconsistent with our research topic. Some short conference papers have a maximum of 4 pages with insufficient description of their proposed approaches or tools, which are of little significance to our work. There are also papers provided by third-party data sources where we cannot confirm the intellectual property rights, such as papers that have not been published in their final version. In addition, the literature collected from different data sources may be duplicated. All the literature needed to be verified and excluded if necessary.

(6) In the process of studying these papers, we further collected literature closely related to our review topic according to the bibliography listed in these papers. Referring to the analysis and summary of systematic literature review methods in Ref. [49], we performed an incomplete backward snowballing from reference lists of the articles identified via our earlier keyword search. Our goal was to find additional relevant papers that may have not been located via keywords alone.

This method of collecting literature inevitably has some limitations. Almost all these processes are implemented manually, and thus the collection efficiency is relatively low and there are inevitably some omissions. Even if the method of a keywords search combined with backward snowballing is used to collect literature, it is difficult to guarantee that the search is exhaustive. The purpose of this review is to collect as much relevant literature as possible, and to give an overview of the research field by analyzing a certain number of papers, especially recently published papers. Therefore, we believe that the existing literature collection is sufficient to support the review work of this paper. Although it would be possible to collect more literature by expanding the search scope and keywords, the relevance of the articles retrieved would be reduced.

III. OVERVIEW OF ANDROID APPLICATIONS

In this section, we do not expand the description of the background knowledge of the Android architecture, security mechanisms, and malware because this has been analyzed and reported in detail in many articles and open source communities [37], [38], [50]. For more Android background details, readers may refer to the references listed in this section.

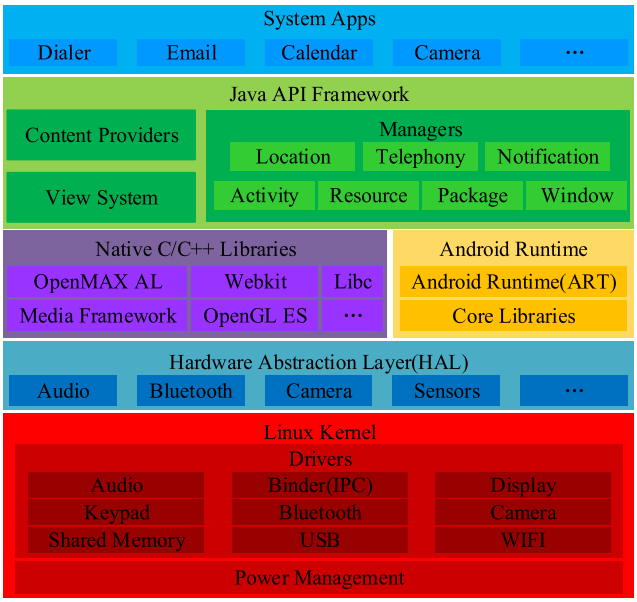

FIGURE 1. Android platform architecture [51].

A. ANDROID SYSTEM ARCHITECTURE

Based on the Linux kernel, the Android operating system adopts a software stack to build its hierarchical system architecture. Google provides the classic layered architecture of the Android system as shown in Fig. 1 [51], which is arranged from bottom to top as follows: Linux kernel, hardware abstraction layer, native C++/C libraries and Android Runtime environment, Java API framework, and the application layer. Each layer contains many submodules and subsystems. The kernel space at the bottom of the Android stack has the Linux kernel as its cornerstone while the user space at the top of the Android system is composed of native C++/C libraries, Android Runtime, and the Java API framework. The kernel and user spaces are connected by system calls. User space programs are mainly written in C++ or Java. Through the Java native interface, the Java layer and the native layer of the user space are connected to the rest of the Android system [52].

B. ANDROID SECURITY MECHANISMS

In general, Android is a privilege-separated operating system. The system achieves high-level system functions by performing a set of system services through an interprocess communication mechanism known as Binder. The Android system isolates running applications using their unique system identifiers (Linux UIDs). Android applications are granted very few permissions by default; they must obtain fine-grained permissions to interact with system services, hardware devices, and other applications.

The permissions required by an Android application are defined in the corresponding manifest file (Androidmanifest.xml) and are granted when the application is installed or while running. The Android system uses UIDs to distinguish the permissions granted to each application, executes these permissions while the application is running, and further constrains the permissions of each process with SELINUX [53].

In the process of system update and iteration, Android developers have devoted a great deal of attention to the improvement of security functions. For example, Android Q, which was released in 2019, has several new security features, such as file-based encryption, access control for sensitive information, access control for background camera/microphone, a lock mode, encrypted backup, and a mechanism called Google Play Protect. Android Q protects user privacy and security from multiple perspectives [54]. It also has an improved permission control mechanism, gives users more control over divulging their location, prohibits background applications from starting activities, restricts application access to non-reset device identifiers (such as IMEI and serial number), and enables MAC address randomization by default. Nevertheless, malware is still a problem.

C. CLASSIFICATION OF ANDROID MALWARE

Malware is a type of application that contains malicious executable code that can destroy the normal or preset services and functions of a system or other application [55]–[58]. Reference [59] classifies malware on smart devices from three perspectives: attack goals and behavior, distribution and infection routes, and privilege acquisition modes. Attack goals and behavior may include, but are not limited to, fraud and service misuse, spamming, espionage, data theft, and sabotage. Distribution and infection pathways include the software market, applications, web browsers, SMS, network, and PCs, among others. Methods of privilege acquisition include user manipulation and technical exploitation.

With the rapid development of mobile internet and smart devices, malware targeting the Android mobile platform has emerged in various forms. Referring to the traditional concept of malware, Android malware can be divided into the following: trojan, backdoor, worm, botnet, spyware, aggressive adware, and ransomware [38]. Felt *et al.* [60] classified Android malware according to human behavioral motivations, including novelty and amusement, selling user information, stealing user credentials, making premium-rate calls, sending SMS messages, SMS spam, search engine optimization, and obtaining ransom. Zhou and Jiang [61] classified and analyzed Android malware from the perspective of malware installation, activation, malicious payloads, and permission abuse. In its research reports, Google [62] uses very conservative words to describe malware, such as potentially harmful applications (PHAs). The PHAs in the Google application market are classified as click fraud, SMS fraud, spyware, toll fraud, trojans, hostile downloaders, backdoors, phishing, privilege escalation, and commercial spyware.

IV. ANDROID MALWARE DETECTION APPROACHES BASED ON MACHINE LEARNING

Machine learning is a branch of artificial intelligence research and application, and several researchers have provided definitions of machine learning [63], [64]. According to [65], machine learning consists of a range of techniques for automating the making of predictions based on past observations. Based on an analogy between machine learning algorithms and the tasks performed by the human brain, machine learning can be roughly divided into five paradigms with different theoretical ideas: symbolists, connectionists, evolutionaries, Bayesians, and analogizers. Each category of machine learning has its own research areas and corresponding algorithms, according to their respective fundamental concepts [66]–[68]. Another representative and widely used classification of machine learning approaches is based on the learning method, which is typically divided into supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning [69]–[72].

Supervised learning makes use of a labeled dataset of samples or instances to train the predictive model, which is often used to solve classification or regression problems. When the prediction of the output is a continuous variable, it is a regression problem, and when the prediction is discrete, it is a classification problem.

Unsupervised learning does not require specially labeled datasets to train the prediction model. The purpose of this kind of machine learning is to discover the internal structure or distribution characteristics of the datasets themselves, and it is often applied to problems such as data clustering and feature dimension reduction.

Semi-supervised learning combines elements of supervised learning and unsupervised learning, using both labeled and unlabeled data. The basic idea of semi-supervised learning is to enable a learner to label the unlabeled sample data by using a model of the data distribution [73]. This type of machine learning is primarily used in scenarios where there is only a small amount of labeled data in the dataset.

In reinforcement learning, there is no labeled data as in supervised learning. Reinforcement learning proceeds as a cycle of prediction and evaluation, where the input data is transferred directly into the model, leading to the dynamic adjustment of the model parameters. The learning model and training data are inferred by receiving feedback from the environment to update the model parameters [74]. Common applications of this type of machine learning include dynamic systems and robot control.

A typical project that uses machine learning methods to solve real-world problems consists of the following main processes [75]–[78]:

(1) Abstract the problem to be solved: Establish whether it is a classification problem, a regression problem, a clustering problem, or something else.

(2) Sample data acquisition and analysis: The acquired data should be representative of the problem domain. It should be of sufficient volume and not be excessively skewed.

(3) Data preprocessing: In view of the limitations and potential errors within sample data, the optimization is realized through preprocessing. This includes data cleaning, normalization, discretization, factorization, missing value processing, dataset segmentation, and other methods.

(4) Feature selection: This involves selecting the most significant features and discarding insignificant features using relevant techniques of feature validity analysis, such as correlation analysis, the chi-square test, average mutual information, conditional entropy, and posterior probability.

(5) Model selection and training: Select the model according to the data and the problem to be solved and use the training data to obtain the model parameters.

(6) Model evaluation and optimization: Use test data to evaluate the model in terms of accuracy, training speed, state space complexity, reliability, portability, and generalization, and optimize the model with the appropriate methods.

(7) Use the new dataset to make predictions and solve practical problems.

(8) Evaluation of the machine learning method to learn its performance (e.g., accuracy, specificity) on the new dataset.

Traditional Android malware detection relies on a library of malicious code features, which needs to be updated over time to ensure the accuracy of detection results. In recent years, researchers have widely applied machine learning techniques to detect malware through training models on a large number of features to achieve the capability of detecting new malware. Current theoretical and practical work mainly focuses on sample acquisition, data preprocessing, feature selection, machine learning models and theory, evaluation of detection effectiveness, and other aspects.

Based on the current literature, the following sections of this paper analyze and summarize the above key aspects of Android malware detection based on machine learning.

A. SAMPLE ACQUISITION

It is essential to train the model with good samples of data so that the acquired model can be applied reliably to make predictions on new data [79]. The obtained sample data should be representative and adequate; otherwise, it may lead to misleading conclusions. For a classification problem like the detection of Android malware, the sample data should not be skewed too much in terms of the number of instances of benign and malicious applications.

The method of obtaining samples of benign Android applications is relatively straightforward. Since the Android applications that are available in various app stores are generally subject to strict testing before they are released, app stores can be expected to be a good source of benign applications. Most of the relevant research uses the method of crawling applications from the mainstream app market such as Google Play to obtain benign samples [6], [80]. A slight variation is that some studies crawl applications with high rating scores and a large number of downloads from the app store [81], or use tools such as VirusTotal [82] and AndroBugs [83] to further ensure that their samples are benign [84].

Compared to obtaining benign applications, there are more ways to obtain samples of Android malware. Some studies have used VirusShare [85], Contagio [86], and other websites that share malware to obtain samples. However, these samples are often non-standardized and will lack metadata such as descriptions, user ratings, and download numbers, which hampers the scope of the subsequent analysis. With the development of Android malware detection research, some specialized malware sample libraries have been created. One representative dataset is MalGenome [61], which is part of the Android Malware Genome Project and contains 1,260 applications from 49 different malware families. The dataset from the Drebin project [87] contains 5,560 applications from 179 different malware families. MalGenome, Drebin, and other datasets of Android malware have been widely used by several researchers. However, as the evolution of Android malware continues, datasets created just a few years ago quickly become outdated.

In recent years, several research organizations and personnel have been working to build and update a larger and more standardized sample library of Android applications. Taking the AndroZoo [88] project as an example, it has collected more than 10 million Android applications and more than 20 kinds of metadata related to each application, including the size of the application, hash values, and permission list in the Androidmanifest.xml file, as well as VirusTotal's report on each application. Based on the AndroZoo dataset and combined with machine learning methods, Ref. [89] achieved good results in the detection of malware. Reference [90] made comprehensive use of VirusShare, MalGenome, AndroZoo, and other Android application market resources to form a sample library containing 19,725 malicious samples and 10,000 benign samples. Some researchers further verified the distribution and effectiveness of samples after preliminary establishment of the datasets, to ensure the rationality of the sample data. Reference [91] used DroidKin [92] to check the dataset to ensure the uniqueness and representativeness of samples. Meanwhile, Ref. [93] used a resampling technique to address an unbalanced dataset. Recently, researchers have created a database named RmvDroid [94], which claimed to be the first large-scale and reliable Android malware dataset. This dataset contained 9,133 samples, with their metadata (e.g., app description, app ratings) belonging, with high confidence, to 56 malware families.

B. DATA PREPROCESSING AND FEATURE SELECTION

When using machine learning algorithms to detect Android malware, an excellent feature set is vital for training the machine learning algorithm [95]. Many factors need to be considered in forming the feature set from Android applications. For instance, the features should be sufficiently differentiated in the dataset to distinguish between malicious and benign Android applications. There is a lot of work involved

in this step, among which data preprocessing and feature selection are two very important tasks.

When studying and summarizing existing research on Android malware detection based on machine learning, we found that some research papers would include a section on data preprocessing and feature selection, or have two separate sections to illustrate these two aspects of the work. We think it is necessary to first explain the relationship between data preprocessing and feature selection at this point to avoid ambiguity. As a matter of fact, in the process of machine learning, data preprocessing covers more than feature selection. Feature selection is a component of data preprocessing, where its main purpose is to improve the effectiveness of machine learning by removing irrelevant or redundant features [96].

1) DATA PREPROCESSING

Feature extraction based on a dataset can obtain raw feature data, but this data may not be satisfactory due to problems such as inconsistent specifications, redundancy, missing values, and an imbalanced distribution [97]. Machine learning on such raw feature data may be unreliable, and therefore it is vital to carry out data preprocessing, which is a key step that underpins the effectiveness of machine learning [98]. The techniques of data preprocessing include data cleaning, data integration, data reduction, and data transformation [96]:

(1) Data cleaning: This includes addressing noise and correcting inconsistencies in the data using techniques for data smoothing and dealing with missing values.

(2) Data integration: This involves merging data from multiple sources, removal of redundant data, and correlation analysis.

(3) Data reduction: The aim of this stage is to obtain a reduced representation of the dataset that is much smaller in size than the original raw data. Wavelet transform, principal component analysis, clustering, and sampling are all commonly used methods, as is feature selection.

(4) Data transformation: This stage focuses on transforming or consolidating the data into forms more appropriate for further processing. Common methods include normalization, discretization, and aggregation.

Each element of data preprocessing is associated with a range of techniques, which are well summarized in Ref. [96] and therefore not be repeated in this paper. In research on the detection of Android malware based on machine learning, the techniques adopted in data preprocessing are relatively standard due to the limited types and styles of the original datasets. We summarize some references on data preprocessing in the detection of Android malware based on machine learning in Table 2. It should be noted that some of the literature is omitted from Table 2 due to a lack of detailed description of data preprocessing in those articles. In addition, if the description of data preprocessing mainly focuses on feature selection, those studies are summarized in subsection IV-B2. We classify the references listed in Table 2 according to the four types proposed in [96].

In the data cleaning phase of preprocessing, some researchers remove unnecessary features based on subjective judgment from a macro perspective, such as Refs. [93] and [101]. Meanwhile, some researchers clean up the data according to the requirements of subsequent processing steps. For example, [102] deletes data such as email address, URL links, punctuation, and stop words, according to the requirements of subsequent natural language processing.

In the data integration phase, researchers mainly form feature datasets for subsequent machine learning by drawing on multiple types of data. For example, Ref. [103] integrates basic information on equipment, a list of installed applications, system calls, and other information into feature vectors. Furthermore, Ref. [104] integrates 6 types of data to form feature datasets that represent the complexity of the application.

In the data reduction phase, in addition to feature selection, which is covered in subsection IV-B2, some researchers directly adopt a method of feature identification substitution to reduce the dimensionality of feature vectors, such as Ref. [106]. Other studies reduce the feature space into a new one composed of a linear combination of the principal components of the raw data by means of principal component analysis (PCA) (for example, Ref. [107]). Meanwhile, some studies use data clustering to represent the original data, such as Ref. [108].

There are various methods involved in the data transformation phase of preprocessing. For example, Refs. [109]–[111] convert binary files into standard images and then generate the data format required for further processing by combining normalization and other methods. In Refs. [112]–[116], the n-gram model from natural language processing is used to represent Dalvik instructions, API calls, and other data. Additionally, in Ref. [119], a one-dimensional feature vector is converted into a two-dimensional matrix to facilitate deep neural network learning.

2) FEATURE SELECTION

Feature selection is a common method of dimensionality reduction. By eliminating redundant and irrelevant features, the size of the dataset can be significantly reduced. The aim of feature selection is to select the optimal subset of features to improve the generalization performance and operational efficiency of machine learning. In general, feature selection involves four basic steps [121]:

(1) Generation: generate the candidate subset of features;

(2) Evaluation: evaluate the quality of the feature subset;

(3) Stopping criterion: decide when to stop the generation procedure;

(4) Validation: check whether the feature subset is valid.

Each of the above steps has a series of principles and methods, which are well summarized and analyzed in Ref. [121]. According to the degree of integration with the machine

TABLE 2. Summary of data preprocessing in selected references related to Android malware detection based on machine learning.

Year	Reference	Description	Type of Data Preprocessing
2012	[99]	Control flow graphs (CFGs) with no more than 5 nodes are discarded because such CFGs are relatively rare and contain less information, and thus discarding them can greatly improve processing speed.	Data Cleaning
2014	[100]	Natural language processing (NLP) is used to filter and extract topic descriptions of the application. Applications with less than 10 topic words and without calling any sensitive APIs are removed, leaving 22,521 applications for subsequent clustering and analysis.	
2017	[101]	System calls that are never invoked by any application in the dataset are removed.	
2018	[93]	There are a lot of ambiguous vectors in the eigenvector matrix, which will affect the performance of the classifier. By removing these ambiguous vectors, the effect of classification can be improved.	
2019	[102]	Preprocess the application descriptions in the app store and then subject them to topic analysis. There are three phases to this process: (1) use regular expressions to remove non-text descriptions; (2) tokenize the description into a list of words, then remove punctuation and stop words; (3) reduce words to their root.	
2011	[103]	Basic information of the device, the list of installed applications, system calls, and other information are integrated into the feature vectors for subsequent analysis.	Data Integration
2014	[104]	The feature data that measures the complexity of a single method or class is aggregated through six functions to form a complete feature set of application complexity.	
2019	[105]	API information is extracted from a control flow graph (CFG); three types of data on the API are established and integrated into the feature set. The three types of data are API usage data, API frequency data, and API sequence data.	
2015	[106]	To reduce the dimensionality of feature vectors and avoid overfitting, features are kept as generic as possible by replacing specific identifiers of the application with tokens.	Data Reduction
2017	[107]	Principal component analysis (PCA) is used to reduce the feature space to a new feature space composed of a linear combination of components from the original features.	
2018	[108]	The affinity propagation (AP) clustering algorithm is used to replace the original data with a clustered representation of the data to reduce feature size.	
2016	[109]	The binary executables are disassembled into opcode sequences, and then converted into images. Histogram normalization and other methods are used to enhance the contrast between malicious and benign application images.	Data Transformation
2017	[110]	Researchers extract the APK file, convert the 4 files of classes.dex, AndroidManifest.xml, resources.arsc, and CERT.RSA4 into 8-bit unsigned integer vectors, organize them into a two-dimensional array, and finally visualize them as grayscale images. The grayscale images are decomposed by wavelet transformation, and image textures can be obtained for subsequent machine learning.	
2018	[111]	The binary files are used to generate grayscale images, and then a bilinear interpolation algorithm is used to preprocess the grayscale images so that the images have the same length and width, thereby making them suitable for the subsequent image classification by a convolutional neural network (CNN).	
2014	[112]	The API call sequences at runtime are extracted, represented in the form of n-grams, and finally normalized.	
2015	[113]	Dalvik instructions are represented in the form of n-grams, and the frequency of different n-grams is calculated for further processing.	
2017	[114]	Dalvik bytecode's n-gram form and the corresponding occurrence frequency are obtained as the feature vector.	
2017	[115]	Dalvik instructions are expressed in the form of n-grams.	
2019	[116]	The API call sequences are represented in the form of n-grams.	
2010	[117]	Raw data such as continuously measured data and events within Android applications are obtained by monitoring. Knowledge-based temporal abstraction (KBTA) is used to transform raw data into time-based features.	
2016	[118]	One type of feature is the co-occurrence matrix vector. The co-occurrence matrix is established based on the system call sequence and is then normalized and finally transformed into a vector.	
2018	[119]	Researchers convert the opcode sequence into a matrix vector, and transform the one-dimensional vector into a two-dimensional matrix, which is suitable for subsequent learning in a deep neural network (DNN).	
2018	[120]	The function call graphs (FCGs) extracted from an APK file are used to generate the topological signatures of the corresponding applications.	

learning algorithm, feature selection methods can be divided into two categories: filter and wrapper. Filtering is independent of the subsequent machine learning algorithm and generally has the characteristics of high efficiency, low complexity, and strong commonality. The wrapper approach directly uses the prediction accuracy of the subsequent machine learning algorithm to evaluate the quality of the generated feature subset. Compared with the filter method, the wrapper method can obtain a better feature subset, but it is more complex and less efficient.

The selection of an evaluation metric is an important factor affecting the performance of feature selection. Dash and Liu [121] divided evaluation metrics into five types: distance metrics, information metrics, dependence metrics, consistency metrics, and classifier error rate metrics.

(1) Distance metrics: Also known as separability or discrimination metrics, distance metrics assess the discriminability between features by calculating the distance between them. The distance can be expressed in terms of geometric distance and probability distance. Geometric distance

TABLE 3. Comparison of different evaluation functions in feature selection.

Evaluation Function	Generality	Calculation Overhead	Classification Accuracy	Applicable Feature Types	Dependence on Machine Learning Algorithm	Feature Selection Algorithm or Evaluation Index
Distance Measure	Strong	Low	Uncertain	Continuous/Discrete	Filter	Absolute value distance, Euclidean distance, Chebyshev distance, Kolmogorov distance. Relief algorithm, Relief-F algorithm.
Information Measure	Strong	Low	Uncertain	Continuous/Discrete	Filter	Information gain (Mutual information). BIF algorithm, MDLM algorithm.
Dependence Measure	Strong	Low	Uncertain	Continuous/Discrete	Filter	Chi-square statistics, T-test, Pearson correlation coefficient, Fisher score. POE1ACC algorithm, PRESET algorithm.
Consistency Measure	Strong	Low	Uncertain	Discrete	Filter	Focus algorithm, LVF algorithm.
Classifier Error Rate Measure	Weak	High	High	Continuous/Discrete	Wrapper	SFS algorithm, SBS algorithm, LVW algorithm.

refers to a distance in geometric space, such as absolute value distance, Euclidean distance, LAN distance, Mahalanobis distance, and Chebyshev distance. Probability distance is used to measure the discriminability between features from the perspective of probability, and an example is the Kolmogorov distance [122]. Relief [123] and Relief-F [124] are typical feature selection algorithms using distance as the evaluation metric.

(2) Information metrics: These apply the concept of information entropy to feature selection. Information metrics use information gain (mutual information) and other quantitative indexes of features to make the feature selection. The information gain of a feature is defined as the difference between the prior uncertainty and expected posterior uncertainty when including this feature. BIF [125] and MDLM [126] are typical feature selection algorithms that use information as the evaluation metric.

(3) Dependence metrics: Also known as correlation metrics, dependence metrics are used to evaluate the degree of correlation between objects. The correlation coefficient is a popular evaluation index used to evaluate the degree of linear correlation as a real value between -1 and 1. The closer the absolute value of the correlation coefficient is to 1, the stronger the correlation; meanwhile, the closer it is to 0, the weaker the correlation [127]. Commonly used correlation metrics include Chi-square statistics, T-test, Pearson correlation coefficient, and the Fisher score. POE1ACC [128] and PRESET [129] are typical feature selection algorithms that use dependence as the evaluation metric.

(4) Consistency metrics: What the distance, information, and dependence metrics mentioned above have in common is that they try to find the feature set that can best help the classifier distinguish between the options to be predicted [130]. From another point of view, a consistency metric tries to find the smallest feature subset that has the same ability to judge the options to be predicted as the original feature set. Consistency metrics can find a smaller subset of features by eliminating irrelevant and redundant features, but they are only applicable to discrete features and are greatly affected by noisy data. Focus [131] and LVF [132] are typical feature selection algorithms using consistency as the evaluation metric.

(5) Classifier error rate metrics: Classifier error rate metrics evaluate the feature subset using the classifier itself. They use several candidate subsets to train the classification model, and the subset with the minimum classification error is deemed the best feature subset. Feature selection using this type of evaluation function is a type of wrapper method. SFS, SBS [133], and LVW [134] are typical feature selection algorithms using classifier error rate metrics as the evaluation function.

In Table 3, we compare the above five types of evaluation metrics from the aspects of generality, calculation overhead, classification accuracy, applicable feature types, dependence on machine learning algorithm, and typical feature selection algorithm or evaluation index.

According to our literature review, we present a summary of the feature selection algorithms and evaluation indexes in Android malware detection based on machine learning in Table 4. In Refs. [108], [114], and [135]–[146] in Table 4, only the information gain (mutual information) is selected as the index to evaluate the generated feature subset. Information measures are a non-parametric and non-linear evaluation standard that do not depend on the distribution of the sample data, and therefore they are widely used in feature selection. In addition, some researchers integrate multiple evaluation indexes to select the feature subset, such as Refs. [101] and [147]–[149]. The genetic search (GS) is a search method based on the genetic algorithm (GA) [157], [158]. Researchers in Ref. [155] claim that it is the first time that the genetic search (GS) was used to select features in Android malware detection. Some studies make use of previous research experience to select the feature subset. In Ref. [30], the method of feature selection based on empirical knowledge was categorized as ''selection based on rationalizing''. For example, in the selection of static features, Ref. [156] refers to the experience of Arp *et al.* [87] in the development of Drebin.

TABLE 4. Summary of feature selection in selected references related to Android malware detection based on machine learning.

Year	Reference	Feature Selection Algorithm or Evaluation Index
2013	[135]	Information gain (Mutual information)
2014	[136]	Information gain (Mutual information)
2015	[137]	Information gain (Mutual information)
2015	[138]	Information gain (Mutual information)
2015	[139]	Information gain (Mutual information)
2015	[140]	Information gain (Mutual information)
2017	[114]	Information gain (Mutual information)
2017	[141]	Information gain (Mutual information)
2017	[142]	Information gain (Mutual information)
2017	[143]	Information gain (Mutual information)
2018	[108]	Information gain (Mutual information)
2018	[144]	Information gain (Mutual information)
2018	[145]	Information gain (Mutual information)
2019	[146]	Information gain (Mutual information)
2012	[147]	Information gain (Mutual information), Chi-square statistics (CS), Fisher score (FS)
2014	[148]	Information gain (Mutual information), Dependence measure
2017	[101]	Information gain (Mutual information), Dependence measure
2018	[149]	TF-IDF, cosine similarity
2014	[150]	Sequential forward selection (SFS)
2015	[106]	Fisher score (FS)
2016	[151]	Dependence measure
2016	[152]	Pearson correlation coefficient
2016	[153]	Chi-square statistics (CS)
2017	[154]	Relief algorithm
2018	[155]	Genetic search (GS)
2018	[156]	The selection of static features refers to the experience of Arp et al. in the Drebin project.

C. FEATURE TYPE

This subsection summarizes and analyzes the features selected by various machine learning algorithms in the field of Android malware detection. These features can be arranged into three categories: static features, dynamic features, and hybrid features, depending on whether they are acquired by running an Android application [6], [17], [23], [24]. The analysis methods used to obtain these three types of features are called static analysis, dynamic analysis, and hybrid analysis, respectively.

The static analysis method analyzes the application and related objects without executing the application [35]. Most static methods use techniques that parse program source code to traverse program paths to check some properties [36]. After the application package (APK) file is decompressed, many of the analysis objects used in the static method can be extracted, such as the AndroidManifest.xml file, which describes permissions, API calls, package name, referenced libraries, and application components (e.g., activities, services). Another example is the classes.dex file, which contains all Android classes compiled into dex file format [23]. Some static methods may represent the analyzed application code as an abstract model (e.g., the opcode in the form of n-grams) based on the purpose of the research. Other information about the application, such as the metadata (e.g., app description, app ratings, app download numbers), can be collected for static analysis from other perspectives.

The dynamic analysis method analyzes features while the application is running (on a real device or virtual environment) [35]. Dynamic analysis is mainly used to identify the behavior characteristics of an Android application, and techniques such as function call monitoring, information flow tracking, and instruction tracing can be applied [17], [24]. The objects of dynamic analysis are network traffic, battery usage, CPU utilization, IP address, and opcode, among others. One type of dynamic analysis relies on the Dalvik runtime (or ART runtime) to obtain the same level of privileges as the Android application, which typically requires modifications to the operating system or the Dalvik virtual machine. Another type of dynamic analysis generally uses emulators and virtual environments for data collection and analysis and achieves higher security through isolation [23].

The hybrid analysis method is a comprehensive approach gaining the benefits of static analysis and dynamic analysis. It combines the two methods in different forms. A hybrid approach provides a better balance between resource and time efficiency, code coverage, method robustness, detection accuracy, and depth [23].

1) STATIC FEATURES

Features obtained by analyzing the source code or other information associated with the application are called static features [159], [160], and the corresponding method of analysis is called static analysis. Specifically, for Android applications, the main object of analysis is the APK file, which is the Android application installation package. Files including AndroidManifest.xml, smali files, etc., can be obtained by decompiling APK files. Further analysis of these files reveals a set of static features, including permissions, API calls, Dalvik opcodes, and other components. According to our review, we summarize the static features used in Android malware detection based on machine learning in Table 5. It should be noted that we chose representative, highly cited, or recently published literature as examples for Table 5.

According to Table 5, we find that when researchers detect malware using only one kind of static feature, they usually select either permission (rows 1–7), API call (rows 8–20), opcode sequence (rows 21–29), or function call graphs (rows 30–32) as features. This phenomenon reflects the close correlation between these static features and whether Android applications contain malware. Moreover, this close correlation is consistent with the conclusion that ''Android permissions are the best single predictor of the app's malignity'' in Ref. [14]. There are also some studies that select or extract static features from other perspectives. Reference [104] detects Android malware from the perspective of software complexity. It extracts 144 features that reflect the complexity of a program's control flow, data flow, and object-oriented design. Meanwhile, Ref. [176] defines a kind of static feature named a modality vector, which is generated in three steps: behavior graph generation,

TABLE 5. Summary of the static features used in selected references related to Android malware detection based on machine learning.

No.	Year	Reference	Features	No.	Year	Reference	Features
1	2012	[161]	Permission	34	2014	[176]	Modality vector (MV)
2	2013	[162]	Permission	35	2016	[151]	Inter-component communication (ICC)
3	2013	[163]	Permission	36	2017	[110]	Texture features from the grayscale image
4	2014	[150]	Permission	37	2017	[177]	Third-party calls
5	2015	[140]	Permission	38	2019	[146]	Number of objects
6	2015	[164]	Permission	39	2013	[178]	Permission, API call
7	2018	[3]	Permission	40	2014	[148]	Permission, API call
8	2013	[165]	API call	41	2015	[138]	Permission, API call
9	2015	[166]	API call	42	2015	[179]	Permission, API call
10	2016	[152]	API call	43	2017	[180]	Permission, API call
11	2016	[167]	API call	44	2017	[181]	Permission, API call
12	2017	[107]	API call	45	2017	[182]	Permission, API call
13	2017	[168]	API call	46	2017	[143]	Permission, API call
14	2017	[169]	API call	47	2018	[183]	Permission, API call
15	2018	[144]	API call	48	2012	[99]	Permission, Control flow graph (CFG)
16	2018	[170]	API call	49	2012	[184]	Permission, Component, Intent, API call
17	2018	[145]	API call	50	2013	[185]	Permission, Intent filters, Native code, Zip files
18	2019	[105]	API call	51	2013	[186]	Permission, Other features of the manifest file (Uses-feature tag)
19	2019	[89]	API call	52	2014	[87]	Permission, API call, Intent, Component, Network Address, etc. (About 545,000 features)
20	2019	[116]	API call	53	2014	[187]	Intent, Permission
21	2013	[135]	Opcode sequence	54	2014	[100]	Description in the app store, API
22	2014	[136]	Opcode sequence	55	2015	[137]	Permission, API call, Specific Linux command
23	2015	[113]	Opcode sequence	56	2016	[153]	Permission, Component
24	2016	[109]	Opcode sequence	57	2016	[188]	Permission, Intent, System command, Suspicious API call, Malicious activity
25	2017	[171]	Opcode sequence	58	2017	[189]	Class-level dependence graph (CDG), Method-level call graph (MCG)
26	2017	[114]	Opcode sequence	59	2018	[111]	Grayscale image, Opcode sequence
27	2017	[172]	Opcode sequence	60	2018	[149]	Permission, API call, System event, URL
28	2018	[108]	Opcode sequence	61	2018	[155]	Permission, Code-based features, Directory path
29	2018	[173]	Opcode sequence	62	2018	[190]	Hardware components, Requested permission, Component, Filtered intent, Restricted API call, Used permission, Suspicious API call, Network address
30	2013	[174]	Function call graph (FCG)	63	2018	[191]	(1) String features: Permission, Hardware feature, Filter intent, Restricted API call, Used permission, Code pattern. (2) Structural features: Function call graph (FCG)
31	2017	[175]	Function call graph (FCG)	64	2018	[192]	Permission, API, and other key application information such as Dynamic code, Reflection code, Native code, Cryptographic code, etc.
32	2018	[120]	Function call graph (FCG)	65	2019	[102]	Description of function, Data flow, Permission
33	2014	[104]	Software complexity	66	2019	[193]	Permission, Hardware feature

sensitive node extraction, and modality generation. Reference [151] conducts malware detection from the perspective of inter-component communication (ICC) among Android applications. It extracts features associated with ICC from four types of objects: components, explicit intents, implicit intents, and intent filters. Reference [110] carries out malware detection from the perspective of textural features within grayscale images. It first unzips the APK file, then converts the files classes.dex, AndroidManifest.xml, resources.arsc, and cert.rsa into 8-bit unsigned integer vectors, and finally converts them into grayscale images for further processing. Reference [177] focuses on the static features of third-party API calls, which are difficult to obfuscate and therefore can improve the detection accuracy. Lastly, Ref. [146] establishes the static feature set from the perspective of quantity, including the number of multiple objects such as lines of code (loc), permissions, and activities.

In Table 5, we find that a lot of Android malware detection research is based on a combination of multiple static features (rows 39–66), most of which are based on a combination of features including permissions or API calls, or a combination of only two types of these features (rows 39–47). To some extent, this reflects the important role these two types of static features have in the detection of malware. Considering that different features have a different influence on detection results, Ref. [191] divides the static features used into string features and structural features, and assigns a weight of 60% to predictions based on string features and 40% to predictions based on structural features. The researchers in Ref. [189] divide the detection of Android malware into two stages. Each stage is based on a different kind of static feature, namely, coarse-grained class-level dependence graphs (CDG), and fine-grained method-level call graphs (MCG). Finally, the researchers in Refs. [100] and [102] generate a class of

static feature by analyzing the description of functions of the application in the app store, and check whether the behavior of the Android application meets its claimed functionality in combination with the features of API calls or data flows.

The acquisition and analysis of static features consumes relatively little time and resources, but the use of code obfuscation, dynamic code loading, and other techniques present significant obstacles for static analysis [4].

Take obfuscation as an example. Through obfuscation techniques, malicious code and all its harmful functionality are difficult to detect and understand by static analysis until they are activated [24]. In Ref. [23], researchers divide obfuscation technology into three categories: trivial transformations, transformations hindering static analysis, and transformations preventing static analysis.

Trivial transformations do not require changes at the code or bytecode level, and this type of obfuscation is mainly used to prevent signature-based analysis by methods such as decompressing and repackaging APK files. Transformations hindering static analysis are used for specific static analysis techniques. For example, feature-based analysis is often susceptible to data obfuscation, and structural analysis is generally vulnerable to control flow obfuscation. Data obfuscation modifies APK data, such as renaming application methods and classes, or reordering or encrypting instance variables and strings. Control flow obfuscation confuses the flow of an application by moving method calls or reordering code. Transformations preventing static analysis usually uses more complete bytecode encryption or Java reflection, which renders static analysis methods ineffective.

In addition, dynamic code loading has a great impact on static analysis. Android applications can load JAR files or shared libraries from remote sources at runtime. This facilitates application development while making it difficult for static methods to perform security analysis on loaded or generated code [37].

2) DYNAMIC FEATURES

When running Android applications in real environments or emulation environments such as a sandbox, the acquired runtime behavioral features are known as dynamic features [194], and the corresponding method of analysis is known as dynamic analysis. Specifically, for Android applications, the objects of dynamic analysis include system calls, API calls, network traffic, and CPU data. According to our literature review, we present a summary of the dynamic features used in Android malware detection based on machine learning in Table 6. Many studies use dynamic analysis technology to detect Android malware, such as [195] and [196], which apply dynamic taint analysis, and [197], which uses Dalvik opcode combined with graph theory. However, since these examples are not primarily based on machine learning methods and are thus outside of the scope of this paper, we do not list them in Table 6.

From Table 6, we find that when researchers detect malware using just one kind of dynamic feature, they usually select either system calls (rows 1–6), API calls [112], [200], and network traffic [201]–[203]. This phenomenon reflects the close correlation between these dynamic features and whether Android applications contain malware or not. Even though malware detection is based on a single type of dynamic feature, different researchers have different concerns or analysis perspectives. Taking the feature of system calls as an example, Refs. [118], [139], and [198] analyze the application based on the sequence of system calls when the application is running. Conversely, Refs. [101] and [199] analyze the application based on the frequencies of different system calls.

TABLE 6. Summary of the dynamic features used in selected references related to Android malware detection based on machine learning.

No.	Year	Reference	Features
1	2011	[103]	System call
2	2015	[139]	System call sequence
3	2016	[118]	System call sequence
4	2016	[198]	System call sequence
5	2017	[101]	Frequency of system calls
6	2017	[199]	Frequency of system calls
7	2014	[112]	API call
8	2016	[200]	API call
9	2013	[201]	Network traffic: Connection duration, TCP size, Number of GET/POST parameters
10	2014	[202]	Network traffic: Average packet size, Average traffic duration, Time interval between packets
11	2017	[203]	Network traffic: DNS, HTTP, TCP, Origin-destination
12	2010	[117]	CPU, Network traffic
13	2012	[147]	CPU, Memory, Battery, Network traffic, Keyboard, etc.
14	2013	[204]	CPU, Memory, Binder API, Battery, etc.
15	2015	[205]	API call, System call
16	2016	[206]	System call, Decode Binder communication, Abstracted behavioral patterns
17	2017	[142]	CPU, Memory, Network traffic
18	2019	[207]	Method call, Inter-component communication (ICC) intent

In Table 6, we also find that a lot of Android malware detection research is based on a combination of multiple dynamic features (rows 12–18) — most based on features such as network traffic, CPU data, and system calls. This reflects the important role of these types of dynamic features in the detection of malware. The study described in Ref. [142] evaluates whether the application is malicious according to the resource consumption of the application at runtime, focusing on the consumption of CPU, memory, network, and other resources. Researchers in Ref. [207] group 70 dynamic features used into the dimensions of structure, security, and ICC, and determine whether the application is malicious from these three dimensions.

For Android applications, dynamic analysis has many advantages over static analysis [23], [207], [208]. Due to the event-driven nature of the Android system, many objects

TABLE 7. Summary of the hybrid features used in selected references related to Android malware detection based on machine learning.

No.	Year	Reference	Features
1	2014	[210]	Static: Permission, API Dynamic: Behavior (Service startup, Network data transmission, File loading)
2	2015	[211]	Static: Permission, Intent Dynamic: Behavior (Method call, Network data transmission)
3	2015	[106]	Static: Permission, API, Intent, Java package name, Publisher ID for advertisement library Dynamic: Behavior (File operation, Network operation, Data leakage, Phone event, Dynamically loaded code, Dynamically registered broadcast receiver)
4	2015	[212]	Static: Permission, API, Intent, Hardware, Network address Dynamic: Behavior (File operation), Resource consumption (CPU, Memory)
5	2016	[213]	Static: Permission, API Dynamic: Behavior (System call)
6	2016	[214]	Static: Permission, API Dynamic: Behavior (Network activity, File system access, Interaction with the operating system)
7	2016	[215]	Static: Permission, Intent, Sensitive function Dynamic: Behavior
8	2017	[154]	Static: Permission, API, Intent, Uses-feature, etc. Dynamic: Behavior (Running process, SMS activity), Resource consumption (CPU, Power)
9	2017	[141]	Static: Permission Dynamic: Behavior (System call)
10	2017	[216]	Static: Permission Dynamic: Behavior (Network traffic)
11	2017	[115]	Static: Permission, Opcode, App store information (Rating, Download number, Developer reputation, etc.) Dynamic: Behavior (System call, SMS, Administrator privilege abuse)
12	2017	[217]	Static: Permission Dynamic: Behavior (System function, Sensitive permission, Sensitive API)
13	2018	[93]	Static: Permission, Intent, Hardware feature, Software features, IP address, Advertisement module, System security setting Dynamic: Behavior (Sensitive API, System service, IP address)
14	2018	[218]	Static: Permission, App store information (Rating, Download number, etc.) Dynamic: Behavior (System call, SMS service, Sensitive API, etc.)
15	2018	[156]	Static: Permission, API, Intent, Components, Hardware Dynamic: Behavior (System call)
16	2019	[194]	Static: Permission, API, Intent, Min_sdk Dynamic: Behavior (Service startup, File operation, SMS and phone event, Sensitive data leakage, Network data transmission, etc.)

or events cannot be analyzed through static analysis alone and need to rely on the runtime environment; these include lifecycle callbacks, GUI handling, control flow, and data flow at runtime. Some permissions or APIs declared in Android application code such as AndroidManifest.xml do not necessarily mean that they will be actually executed or invoked. Additionally, since Android version 6.0 (API 23), the Android system has added dynamic permission support, so detection based on static analysis alone may be prone to false positives. Techniques such as code obfuscation and dynamic code loading make it difficult to detect malicious behavior in Android applications through static analysis, but dynamic analysis can overcome these limitations to some extent. Although dynamic analysis has many advantages as mentioned above, it also has the disadvantage of consuming more time and resources than static analysis [154], [209].

3) HYBRID FEATURES

Hybrid features are made up of static features and dynamic features, and the corresponding method of analysis is called hybrid analysis [7], [17], [21]. Hybrid analysis can exploit the advantages of both static and dynamic analysis to meet the detection needs in specific scenarios. According to our literature review, in Table 7, we summarize the hybrid features used in Android malware detection based on machine learning.

From Table 7, we find that when researchers select static features as part of a hybrid approach, they tend to use API or intent as features. The use of permission features is particularly prominent, as all the examples in Table 7 adopt them. This phenomenon is consistent with the conclusion drawn from Table 5, in that permissions are widely used for static analysis. In the dynamic analysis stage of the literature listed in Table 7, researchers mainly select application behavior, such as file operations, network behavior, and data transmission at runtime, as the feature for analysis. In contrast to static analysis, dynamic analysis often focuses on more than just a few features.

The selected hybrid features in Table 7 have much in common, but peculiarities still exist in the selection of features for machine learning. In terms of the static features used in Refs. [115] and [218], besides the features of the Android applications themselves, the researchers make use of third-party auxiliary data on the applications from the app store, such as ranking and number of downloads. In terms of the dynamic features used in Refs. [154] and [212], in addition to

TABLE 8. Classification of common machine learning algorithms based on learning method.

Learning Method	Machine Learning Model or Algorithm	
Supervised Learning	1. Decision Trees 2. Naive Bayesian 3. Linear Model (1) Linear Regression: Ordinary Least Squares Regression (2) Logistic Regression (3) Linear Discriminate Analysis (LDA) 4. K-Nearest Neighbor (KNN) 5. Support Vector Machine (SVM)	1. Neural Network (NN) and Deep Learning (DL) 2. Hidden Markov Model (HMM) 3. Transfer Learning 4. Ensemble Learning: Bagging, Boosting, Random Forest 5. Online Learning
Unsupervised Learning	1. Clustering Algorithms: K-means 2. Principal Component Analysis (PCA) 3. Singular Value Decomposition (SVD) 4. Independent Component Analysis (ICA) 5. A-priori Algorithm 6. Expectation-Maximization (EM)	
Semi-supervised Learning	1. Semi-supervised Learning with Nuclear Norm Regularization (SSL-NNR) 2. Graph Inference Learning (GIL) 3. Laplacian SVM	
Reinforcement Learning	1. Q-Learning 2. Deep Q-Learning Network (DQN) 3. Temporal Difference Learning	

the behavioral features of Android applications, the authors exploit information that can reflect whether the applications are working properly, such as CPU consumption, memory usage, power consumption, and other resources. Researchers in Refs. [115], [154], [211], and [216] divide hybrid analysis into two phases, static analysis and dynamic analysis, where the results of static analysis can be used to guide dynamic analysis and improve the pertinence and coverage of dynamic analysis. There are also some studies that classify and extract features from a special perspective and do not strictly distinguish whether the features used in the research are obtained by static or dynamic analysis. For example, Ref. [93] divides the 377 features used into 10 categories, among which the IP address is derived from both static source code analysis and dynamic behavioral analysis. The researchers in Ref. [218] analyze the application over the four levels, i.e., the kernel level, application level, user level, and package level, and make comprehensive use of both static and dynamic features such as application metadata, API calls, user behavior, SMS services, and system calls.

D. COMMON MACHINE LEARNING MODELS AND ALGORITHMS

1) INTRODUCTION TO COMMON MODELS AND ALGORITHMS

In this subsection, we first classify the commonly used machine learning models and algorithms according to their learning method, as shown in Table 8. There are several points worthy of note:

(1) Methods such as neural networks, ensemble learning, and online learning may be implemented in different ways, such as by applying supervised learning and unsupervised learning, according to the specific situation. Taking neural networks as an example, the perceptron-based algorithm implements supervised learning, while the Boltzmann machine-based algorithm uses unsupervised learning. For another example, ensemble learning combines multiple learners into a predictive model with the aim of improving the accuracy of prediction. Strictly speaking, ensemble learning is not a learning method, but a way of combining learners. Therefore, we put such models or algorithms in the last column of Table 8, rather than categorizing them according to their learning method.

(2) Machine learning techniques can be considered from different perspectives. For example, they can be divided into classification, regression, clustering, and dimension reduction, according to the task objectives. Alternatively, learning can be done via online learning and batch learning, according to whether incremental learning is carried out. The machine learning algorithms listed in Table 8 are categorized according to their learning method, which is the most commonly used classification [69]–[72], as described at the beginning of Section IV. However, there are still controversies. For example, deep learning can be considered as a more complex extension of neural networks. Even though it is difficult to have a universally agreed upon understanding of the classification of machine learning models and algorithms, we list the most popular models and algorithms in Table 8.

Based on classifying common models and algorithms, we briefly introduce those typically used in the detection of Android malware.

(1) Decision Trees (DT): A decision tree is used to make decisions based on data items held in a tree-like structure. As in a kind of thinking and processing mode analogous those adopted by humans when facing decision problems, its

basic process follows the simple and intuitive ''divide and conquer'' strategy [79], [219].

(2) Naive Bayesian (NB): Naive Bayes methods are a group of supervised learning algorithms based on Bayes' theorem. These methods ''naively'' assume that each pair of features is independent from each other [219], that is, each feature independently influences the prediction results [220].

(3) Linear Model (LM): The linear model is based on a function that predicts results through a linear combination of features, and the optimal solution of parameters in the function is solved by using the training dataset [79]. When the linear model is applied to regression problems, it is known as linear regression. When a linear model is used to solve classification problems, logistic regression can be adopted. Taking binary classification as an example, the main principle of the logistic regression method is to use the logistic function to convert the real value generated by the linear regression model into a value of 0 or 1, corresponding to the two categories to be predicted [79].

(4) Support Vector Machine (SVM): From the perspective of geometry, the principle of the SVM is to find an optimal separating hyperplane that meets the classification requirements so that the hyperplane can separate points in an n-dimensional space ''as far as possible'' while ensuring the accuracy of classification [221], [222]. To satisfy the extreme value requirement of ''as far as possible'', a method such as the Lagrange multiplier can be used [79].

(5) K-Nearest Neighbor (KNN): The strategy of the k-nearest neighbor algorithm is to find the k labeled samples closest to the sample to be classified in the sample space. If most of the k-nearest samples belong to a certain category, then the sample to be classified also belongs to that category. Distance can be measured by a variety of metrics, and Euclidean distance is the most commonly used. This algorithm makes a classification according to the distribution of data in a set of labeled samples [223], [224].

(6) K-means Clustering Algorithm: The aim of k-means clustering algorithm is to classify similar objects into the same cluster. This algorithm partitions cases into k different clusters where the number of clusters is specified by the user. The center of mass of each cluster is represented by the mean of all the objects in the cluster [224]. When calculating the distance between each point and the center of mass, many metrics are available, such as the Euclidean distance [225].

(7) Neural Networks (NN) and Deep Learning (DL): Neural networks simulate how biological neurons interact with the real world [226], and date back to the 1940s [227]. In 1943, the neurophysiologist Warren McCulloch and logician Walter Pitts, inspired by the structure of biological neurons, first proposed an abstract ''M-P'' neuron model [228], whose output value is ''0'' or ''1'', corresponding to the two states of neuronal inhibition and excitation in a biological neural network.

Many of these neurons can be connected in a layered structure to form a neural network. From the perspective of macroscopic mathematical calculation, a neural network is a mathematical model with several parameters. Machine learning models based on neural networks mainly includes two categories. One category is based on the perceptron model [229]. The multilayer perceptron (MP) model is commonly used, and it usually utilizes the error back propagation (BP) algorithm [230] to train the network according to the expected output. The other category is based on the Boltzmann machine [231], which is a random neural network whose output is determined according to a probability distribution. Restricted Boltzmann machine (RBM) models [232] are commonly employed, and they usually use the contrastive divergence (CD) algorithm [233] to train the network according to the training data.

Deep learning models consist of multilayer networks that are used to learn representations of data with multiple levels of abstraction [234]. Deep learning emerged from the development of neural networks, and different studies have different views on the relationship between the two. Ref. [235] argues that deep learning is a type of machine learning that is based on neural networks, while Ref. [236] argues that neural networks are the most commonly used form of deep learning. Due to this controversy, the present paper will not impose a distinction between neural networks and deep learning; instead, the two will be combined for the purpose of discussion.

(8) Ensemble Learning: Ensemble learning achieves better performance at generalization than any single learner by combining multiple learning models. According to the way the learners are arranged, ensemble learning methods can be divided into two categories [237]. One is the serialization method where the learners operate sequentially and there is strong dependency between individual learners, as in boosting [238]. The other is parallelization, where the learners operate simultaneously without strong dependency between individual learners, as in bagging [239] and random forests [240].

(9)Online Learning: Online learning does not utilize the entire training set at any one time. Rather, inputs to the model are processed in batches so that the prediction model is constantly updated with new training data [241]. Online learning is suitable for scenarios where algorithms need to be dynamically adjusted to fit new patterns in the data. Commonly used online learning algorithms include passive aggressive learning and adaptive regularization learning [242].

2) MACHINE LEARNING MODELS AND ALGORITHMS USED IN ANDROID MALWARE DETECTION

It should be noted that the machine learning models and algorithms briefly introduced in subsection IV-D1 are commonly used for the detection of Android malware, and each one has many specific implementation methods. In Table 9, we analyze the advantages and disadvantages of the models and algorithms introduced in subsection IV-D1, and then list more than 100 articles that apply these to detect Android malware. Most of the articles listed in Table 9 have been published in the past five years. The machine learning

TABLE 9. Comparison of machine learning models or algorithms commonly used in Android malware detection.

Model or Algorithm	Advantage	Disadvantage	References
Decision Trees (DT)	Simple to understand and interpret. It can handle samples with missing values or large scale.	Prone to lead to overfitting. It does not support online learning.	[3],[93],[101],[105],[118],[135],[137],[138],[141], [144]-[147],[150],[151],[162],[165],[166],[172],[176], [178],[186],[188],[189],[192],[194],[199],[201],[203], [205],[214]-[217], [246]
Naive Bayesian (NB)	The model can be trained easily and quickly.	Not applicable to situations where the feature variables are correlated. The prior probability needs to be calculated.	[93],[108],[114],[118],[137],[138],[141],[143],[144], [146]-[148],[153],[162],[166],[176],[187],[192],[201], [205],[213]-[215],[217],[243],[246]
Linear Model (LM)	It is the main algorithm in statistics; fast and direct.	The premise of the algorithm is strict. It cannot deal with the high-dimensional features well.	[118],[137],[141],[146],[147],[162],[172], [188],[194],[203],[205],[215],[246]
Support Vector Machine (SVM)	It has advantages in solving small-scale, high-dimensional or non-linear problems.	The overhead in data processing is large. It is sensitive to samples with missing values.	[3],[87],[99]-[102],[104],[106]-[108],[112]-[116],[118], [120],[135],[136],[138],[139],[142],[146],[150]-[152],[154], [156],[161],[165],[166],[168],[174]-[176],[178]-[180],[182], [186],[188],[189],[191],[192],[194],[201],[206],[211]-[215]
K-Nearest Neighbors (KNN)	Easy to realize without parameter estimation. It is suitable for solving multi-classification problems.	Greatly affected by data skew. The computation overhead is relatively large.	[89],[93],[101],[104],[107],[108],[118],[135],[141],[144]-[146],[148],[152],[162],[165],[172],[184],[186],[188],[189], [191],[194],[201],[203],[205],[213]-[215],[218],[243]
K-means	Simple, fast, and easy to implement.	Results are affected by the initial setting. It is sensitive to noise and outliers.	[100],[102],[103],[147],[164],[177]
Neural Network and Deep Learning (NN&DL)	Has high accuracy and strong fault tolerance.	Needs a lot of data for training. Parameter and network topology selection is not easy.	[101],[109],[111],[119],[138],[140],[167],[171],[181],[183], [188],[190],[193],[198],[201],[210],[214],[215],[243],[246]-[249]
Ensemble Learning (EL)	Much more accurate than using a single model.	Overhead is large. It requires a lot of model training and maintenance.	[3],[89],[93],[101],[104],[107],[108],[110],[113],[114],[118], [138],[141],[144],[146], [149]-[152],[162],[169],[170], [172],[176],[178],[185],[188],[189],[191],[192],[194],[199], [203],[204],[205],[207],[214]
Online Learning (OL)	Strong adaptability and good real-time performance. It reduces the threshold requirement of hardware performance.	Some models are not suitable for online learning. It is difficult to find the global optimal solution.	[244],[245],[250],[251]

methods used in some of the literature in Table 9 are improvements based on models or algorithms that are also classified under the corresponding models or algorithms. The relevant research that does not specify the machine learning models or algorithms used are not listed in Table 9. To aid explanation, the literature listed in Table 9 is also shown in Venn diagram form in Fig. 2. Some studies have used multiple machine learning models or algorithms, which correspond to the reference numbers shown on overlapping regions or intersections in Fig. 2. There are also some overlapping areas or intersections without elements, which indicates that there is no literature in Table 9 that makes use of all the machine learning methods simultaneously. Note that the area of each shape in the Venn diagram is not proportional to the number of elements it contains. The Venn diagram only shows the distribution in the application of machine learning methods according to the literature listed in Table 9, and does not indicate the proportion or degree of usage of each learning model or algorithm in the field of Android malware detection as a whole.

We believe that there are three modes by which researchers select machine learning models or algorithms to detect Android malware. The first is to design the scheme based on a single model or algorithm. The selection of model or algorithm is often done based on the sample data, selected features, and application scenarios. The second is to choose different models or algorithms according to the purpose of the research. For example, some researchers choose algorithms with a high prediction accuracy, such as neural networks or deep learning. Computational overhead or hardware performance is not the main factor considered. The third mode is to choose a variety of models or algorithms and compare the advantages and disadvantages of each machine learning method in the particular scenario.

As an example of the first mode, Ref. [87] extracts 545,000 features, including permissions and API calls, then selects an SVM algorithm due to its ability to solve high-dimensional problems in analyzing the relationship between features and malicious behavior of the application. In some studies, the algorithms are based on a typical SVM, which is then improved and optimized. For example, Ref. [168] uses a heterogeneous information network (HIN) to represent the relationship between API calls and constructs a multi-kernel learning algorithm based on an SVM. Reference [116] defines a data flow named Complex-Flows, extracts API call sequences by using data flow analysis tools, such as BlueSeal,

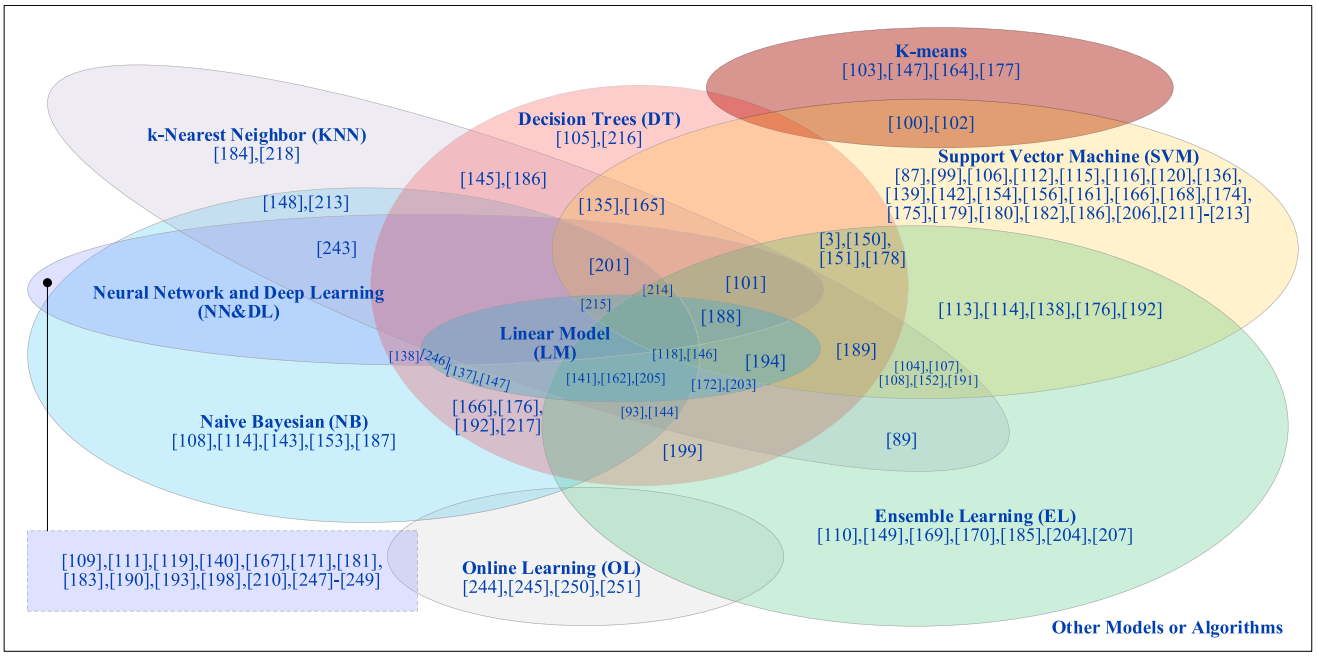

FIGURE 2. Venn diagram of machine learning methods listed in Table 9.

then generates feature sets in the form of n-grams, and finally uses an SVM algorithm to detect malware. In Ref. [100], natural language processing (NLP) technology and the k-means algorithm are firstly used to process the description of each application in the market. Combined with the extracted sensitive API calls, the SVM algorithm is finally used to detect malware. Reference [180] adopts the incremental SVM algorithm to make full use of the prior information of historical samples. It avoids retraining all the data when new samples are added, thus improving detection efficiency.

In the second mode, the purpose or perspective of the research is an important basis for model or algorithm selection. For example, in Ref. [181], an extreme learning machine is used for prediction based on the features of permissions and API calls, with the aim of improving learning efficiency and simplifying the process of setting model parameters. In Refs. [154] and [245], an online passive aggressive algorithm and an online confidence weighted algorithm are used to address the problem of classifier aging. References [107], [198], [200], [246], and [251] apply a Markov model to solve the problem of malware detection from the viewpoint of probability. Furthermore, some researchers optimize the experimental scheme from the aspects of feature construction and data preprocessing. For example, Ref. [167] analyzes the API calls extracted from the smali files of Android applications and group the API calls belonging to a method in the smali code into the same ''API Call Block''. Based on the ''API Call Blocks'', a deep belief network (DBN) is used to detect Android malware. In Ref. [119], the opcode sequence obtained from the Dalvik command in the application is converted into a feature vector, and the one-dimensional vector is further transformed into a two-dimensional matrix so as to facilitate the subsequent deep neural network (DNN) learning. Based on the effectiveness of neural networks in the field of image recognition, Refs. [109] and [111] extract opcodes from the binary executable files and convert them into images. Then neural network algorithms, such as convolutional neural network (CNN) and a long short-term memory (LSTM) network, are used to make predictions.

The third mode is often used in current research on Android malware detection. Here, a range of machine learning models or algorithms are selected for comparative analysis to determine the best classifier for the particular problem. In Ref. [3], the researchers propose a three-layer feature data clipping method. They use this to extract 22 permissions that make the greatest contribution to detection efficiency from 135 permissions, and then carry out a comparative analysis of the detection effectiveness with 67 machine learning algorithms such as a decision tree and an SVM. Reference [89] extracts API calls and then constructs behavioral semantics by association rule analysis to represent the behavioral features of the application. Finally, an SVM, k-nearest neighbor, random forest, and other algorithms are used for detection. Reference [105] extracts three kinds of features of the API from control flow graphs, including APIs (which APIs the suspect application uses), the API frequencies (how many times the application uses APIs), and API sequence (the order the application uses APIs). The authors claim that this is the first time that features have been constructed using the order in which an application uses APIs. Finally, a decision tree, a deep neural network (DNN), and a long short-term memory (LSTM) network are used to detect malware and compare the results.

E. EVALUATION OF DETECTION EFFECTIVENESS

Evaluating the performance of models or algorithms is an important topic in the field of machine learning. Generally, Android malware detection is studied as a typical binary classification problem. The metrics can be used not only to evaluate the predictive accuracy of the classifier, but also to optimize the model. This subsection introduces some evaluation metrics that are commonly used in the field of Android malware detection. Referring to Refs. [79], [252], and [253], we present the content of this subsection in three parts: one is the division of the dataset, the second is the evaluation of classifier performance, and the third is a reliability estimate of the evaluation results.

1) DIVISION OF DATASETS

The original dataset is divided into a training set and a test set. The training set is used to select the model and tune the parameters while the test set is used to evaluate the performance of the classifier. In order to ensure the integrity of the evaluation, the training and test sets should be mutually exclusive, and the distribution of data should be consistent with the original dataset as much as possible. In the evaluation of detection effectiveness, methods such as hold-out, cross validation, and bootstrapping are often used to segment the dataset into training and test sets [79]. Taking the commonly used k-fold cross validation as an example, the main steps are as follows [254]:

(1) The original dataset is divided into k mutually exclusive subsets of roughly equal size.

(2) One of the subsets is selected as the test set, and the remaining (k−1) subsets constitute the training set. The training set is used for model selection and parameter tuning, and then the obtained classifier is evaluated using the test set.

(3) Step 2 is repeated k times to obtain the performance evaluation results for k groups. The average value of the k groups of results is taken as the overall performance of the classifier.

2) EVALUATION OF CLASSIFIER PERFORMANCE

There are many performance metrics for classifiers. A summary can be found in Refs. [255] and [256] while Refs. [257] and [258] propose some new metrics. This section introduces the performance metrics commonly used in the Android malware detection field.

TABLE 10. Confusion matrix of predicted results.

	Predicted Result	
True Class	Positive	Negative
Positive	*TP*	*FN*
Negative	*FP*	*TN*

As a typical binary classification problem, the results of a prediction of whether an Android application contains malware can be divided into four types, as shown through a confusion matrix in Table 10 [259], [260].

The concepts of *FP*, *FN*, *TP*, and *TN* are defined as follows.

(1) True positive (*TP*): the application is a malicious application and was correctly predicted to be malicious;

(2) False positive (*FP*): the application is not a malicious application but was wrongly predicted to be malicious;

(3) True negative (*TN*): the application is not a malicious application and was correctly predicted to be non-malicious;

(4) False negative (*FN*): the application is a malicious application but was wrongly predicted to be non-malicious.

The above four results are mutually exclusive, and thus their sum is the total number of samples in the test. Based on these four basic concepts, a series of performance metrics has been derived. Some commonly used metrics are as follows.

(1) Accuracy (A_{cc}) represents the ratio of correct predictions among the total number of samples in the test. Equation 1 shows how accuracy is computed.

$$A_{cc} = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

(2) Error Rate (E_{rr}) represents the ratio of false predictions among the total number of samples in the test. Equation 2 shows how error rate is computed.

$$E_{rr} = \frac{FP + FN}{TP + TN + FP + FN} \tag{2}$$

(3) Precision (P) represents the ratio of all samples correctly predicted to be positive among all samples predicted to be positive. Equation 3 shows how precision is computed.

$$P = \frac{TP}{TP + FP} \tag{3}$$

(4) Recall (R) represents the ratio of all positive samples correctly predicted among all positive samples. Equation 4 shows how recall is computed.

$$R = \frac{TP}{TP + FN} \tag{4}$$

Precision and recall are very important performance metrics, but they only provide a partial evaluation. In order to combine these two values to obtain a more complete evaluation of the performance of the classifier, the harmonic mean of precision and recall can be used, which is known as the F_1 score. Equation 5 shows how the F_1 score is computed.

$$F_1 = 2 \cdot \frac{P \cdot R}{P + R} \tag{5}$$

A more general form is the weighted harmonic mean of precision and recall, which is denoted by F_β. Equation 6 shows how the F_β score is computed, where β is a positive real number, which means that recall is β times as important as precision.

$$F_\beta = \left(1 + \beta^2\right) \cdot \frac{P \cdot R}{\beta^2 \cdot P + R} \tag{6}$$

Some researchers also evaluate classifier performance through the receiver operating characteristic (ROC) curve [261] and area under the ROC curve (AUC) [262].

This technique is popular because, compared with accuracy and other performance metrics, the AUC is less affected by class imbalance because it is not dependent on the threshold set by the model and is invariant to prior class probabilities [79], [263].

3) RELIABILITY ESTIMATION OF EVALUATION RESULTS

Although we can obtain evaluation results, such as the accuracy of the classifier, by combining dataset and performance metrics, the performance evaluation on a limited dataset is only representative of the classifier's ability to learn patterns within the dataset, rather than its ability to generalize this performance on new data. To what extent should we trust the performance measures of the classifier? In this case, there is a need to conduct reliability estimation of the previous evaluation results of classifier performance. Therefore, answering this question requires the application of statistical hypothesis testing [264], [265], which has a very broad content. For Android malware detection, the test results of each sample can only be benign or malicious, and thus can be approximately represented as Bernoulli trials [266], which have a binomial distribution [267]. Under the condition that the results follow the binomial distribution, we can conduct reliability estimations by using the relatively simple "binomial test" [268]. We briefly introduce some basic principles of the "binomial test" below.

Taking the error rate of the classifier as an example, in reality we only know the test error rate of the classifier on the original dataset, and hence it is impossible to know the generalization error rate ε of the classifier on new data with any degree of accuracy. Hypothesis testing involves making a statistical decision, such as making an assumption that "the generalization error rate ε of the classifier is no higher than ε_0", that is, "$\varepsilon \leq \varepsilon_0$", where ε_0 is a real number that we set between 0 and 1. Hypothesis testing is the process of determining to what extent the hypothesis can be accepted; the basis of this is the null hypothesis, and the hypothesis that differs from the null hypothesis is called the alternative hypothesis [269]. Hypothesis testing sets a significance level (α), which determines whether the hypothesis is to be accepted. The value of the significance level is usually set to 0.05 or 0.1. The confidence corresponding to the significance level is (1-α), which indicates the (1-α) confidence level for the results. The p-value is used to decide whether the null hypothesis is to be accepted based on the sample results. Specifically, if $p < \alpha$, the null hypothesis will be rejected; if $p \geq \alpha$, the null hypothesis will be accepted [270]. For example, if the significance level is $\alpha = 0.05$, $\varepsilon_0 = 0.3$ is a threshold set for the error rate and used as a reference to calculate the p-value. The number of Android applications used for performance evaluation in the test set is N, and the resulting classifier test error rate is ε_t. The process of hypothesis testing is as follows [79]:

(1) Establish the hypothesis and determine the significance level.

Null hypothesis (H_0: $\varepsilon \leq \varepsilon_0$). That is, the generalization error rate of the classifier ε is not higher than ε_0. The corresponding significance level $\alpha = 0.05$.

Alternative hypothesis (H_1: $\varepsilon > \varepsilon_0$). That is, the generalization error rate of the classifier ε is higher than ε_0.

(2) According to the binomial distribution, calculate the p-value.

The number of misclassified applications is $N{\cdot}\varepsilon_0$. In the binomial distribution, the probability that an application is misclassified is the generalization error rate of the classifier ε. Equation 7 shows how the p-value is calculated.

$$P\left(X \leq N \cdot \varepsilon_0\right) = \sum_{i=1}^{N \cdot \varepsilon_0} C_N^i \varepsilon^i \left(1-\varepsilon\right)^{N-i} \tag{7}$$

(3) Only when $p < \alpha$ will the null hypothesis, H_0, be rejected. In other words, we can reject the null hypothesis only if the probability of the observed result occurring by chance is less than our chosen threshold. According to this constraint of the p-value, the extreme value $\bar{\varepsilon}$ of the classifier generalization error rate ε is solved. The test error rate of the classifier ε_t is compared with $\bar{\varepsilon}$, and the conclusion is drawn.

If the test error rate ε_t is less than the extreme value $\bar{\varepsilon}$, the null hypothesis H_0 will not be rejected at the significance level α. In other words, we can take the confidence of $(1-\alpha)$ to believe that the generalization error rate ε of the classifier is not higher than ε_0. If the converse is true, the null hypothesis H_0 will be rejected.

In addition to the binomial test and the cross validation method, a T-test [271], [272] can be used to evaluate the performance of the classifier. When comparing the performance of two classifiers, methods such as a cross validation T-test [273], [274] and the McNemar test [275], [276] can be used. When comparing the performance of multiple classifiers, methods such as the Friedman test [277], [278] and Nemenyi post-hoc test [279], [280] can be useful. These methods are considered in detail in Ref. [79].

The content mentioned in this subsection mainly focuses on evaluation of classifier performance and reliability estimation of the evaluation results. Of course, we could base the evaluation on many other aspects including real-time detection support, preservation of privacy, or economic resource consumption [19]. However, these are outside the scope of our paper and will not be detailed here.

V. RESEARCH DIRECTIONS AND CHALLENGES

Based on the above analysis and summary of the research field of Android malware detection based on machine learning, we believe that there are currently some research directions and challenges. We categorize the content of this section according to different aspects of this research field, as identified in subsections IV-A to IV-E.

A. ESTABLISHMENT OF THE SAMPLE SET

The training and validation of an Android malware detector may produce biased results if the dataset they are based on

is not representative in distribution due to insufficient size or quality. Even if the detector performs well in the experimental tests, it may not work well in the real environment [281]. Therefore, establishing a good dataset of Android applications is an important task.

Due to the open source nature of the Android operating system, users have a number of ways to obtain Android applications. Not only can they download applications from mainstream markets such as Google, Huawei, and Baidu, but they can also obtain applications from many informal third-party markets and websites. Additionally, malicious websites, instant messengers, or storage on the network can share Android applications for users. Currently, as described in subsection IV-A, although many projects such as AndroZoo [88] and RmvDroid [94] have established some standardized Android application sample libraries, there still are some problems in the sample set used for research, such as the relatively outdated and small size of the samples, uneven distribution of malicious and benign samples, and the repackaging of Android applications.

Alternatively, with the aim of accurately balancing the proportion of Android malware in the dataset, the percentage of Android malware in the training set can be adjusted by sampling and other ways so as to improve the predictive effect of the classifier. The experimental analysis under different proportions of Android malware illustrates the influence of sample distribution on the performance of classifier [281], [282]. Although estimates of the proportion of Android malware in specific datasets can be obtained from some studies [283]–[285], and relevant data can also be obtained from the statistical reports of mainstream Android application markets such as Google [62], there is still no strong evidence to show the accuracy of these estimates. In addition, due to the constant development and evolution of Android malware, it is also very important to constantly update the sample set with the latest Android applications. In short, it is an ongoing task to establish a better sample set of Android applications.

B. DATA OPTIMIZATION AND PROCESSING

Data processing runs through the whole process of Android malware detection based on machine learning, including not only the processing of collected Android application samples, but also the processing of the feature set extracted from the samples.

The increasing scale of malware indirectly leads to a huge amount of data to be processed, and thus it is necessary to find effective means to deal with such ''big data''. Meanwhile, the feature data used in some studies is both high-dimensional and large-scale [87], [106]. This trend is likely to become more pronounced as the number of Android applications grows and as the Android framework updates, creating new features for classifiers to learn from, such as new API calls. The processing of high-dimensional mass data will lead to problems such as rising overhead and performance degradation of machine learning models, which are areas where many research fields are striving to make breakthroughs. Distributed or cloud-based architectures can reduce the requirements of data processing on hardware performance to some extent, and methods such as data mining and fusion can also improve the efficiency of big data processing to a certain degree [19].

As described in subsection IV-B, the quality of the feature dataset has a great influence on the effectiveness of machine learning and directly affects the efficiency of malware detection. In the process of feature selection, how to use feature engineering and other methods to evaluate and select feature sets and eliminate redundant and irrelevant features is also a very important research field. Through our literature review, we find that in the field of Android malware detection, there are few studies that focus on feature selection alone as in Ref. [30]. More researchers tend to conduct feature selection based on the traditional practices in the field and pay more attention to the analysis and detection methods of Android malware. By combining the literature on feature selection, we can infer that in the field of Android malware detection, the scalability and stability of the feature selection algorithm is a big challenge. With the rapid growth in the number of Android applications, the feature sets used in many studies are both high-dimensional and large-scale. However, many feature selection algorithms have a much higher time complexity as dimensionality increases, which requires greater scalability of the feature selection algorithms [286]. We hope that the feature selection algorithm can be very stable, that is, less sensitive to perturbations in the feature data. When new samples are added or some samples are deleted, the algorithm should produce a consistent subset of features [95].

In addition, the processing of incompletely labeled data can also be further improved. As new Android applications appear, it is obviously inefficient to classify and label them one by one, and the semi-supervised learning method to deal with incompletely labeled data will be restricted by factors such as efficiency and sample distribution [287]. Some researchers use ensemble learning to label these new applications by combining the weights of multiple learners and their predictions to indirectly solve the problem of incompletely and unlabeled data [288]. However, there will be some problems in model selection and weight assignment, and even the risk that all the learners in the ensemble will make wrong predictions. How to make better use of this unlabeled data to aid machine learning is a challenging task.

C. FEATURE EXTRACTION AND ESTABLISHMENT

How to extract the features of Android applications and which features to extract is a very extensive research topic. The impact of features on machine learning effectiveness and efficiency should be considered comprehensively. Additionally, some malware is designed to deliberately defend against the analysis of researchers or anti-virus tools, which creates an endless game between the two sides.

As mentioned in subsection IV-C, the analysis of Android applications can be divided into three categories: static, dynamic, and hybrid. Different detection methods based on

machine learning may extract different features, which need to be realized by the comprehensive use of different methods. The developers of malware will take a variety of countermeasures in response to these analysis and detection methods. However, due to the characteristics of the Android architecture itself, many analysis techniques will be limited. Static analysis cannot resist code obfuscation, dynamic code loading, and other technologies. There are also many challenges to feature extraction with dynamic analysis. For example, how to realize more accurate behavioral simulation through an emulator that is not convenient to monitor through the application, how to ensure that all malicious behaviors can be triggered during dynamic analysis, and how to use real mobile devices to efficiently generate large-scale dynamic features are all problems to be solved. At the same time, to achieve high code and path coverage, we usually use automation tools for testing to improve the efficiency of the analysis. In this case, time bombs, logic bombs, and login interfaces based on passwords, for example, will bring difficulties to our automated analysis. Although a lot of research work has put forward methods to address these deficiencies, it is still difficult to achieve a perfect unity between the analysis effectiveness and efficiency [4], [35], [37].

Another aspect is to search for new features with a strong training effect. Inter-component communication among Android applications, the descriptions, and user ratings of the applications in the app store are receiving increased attention from researchers. Indeed, in the absence of a revolutionary change in the architecture of the Android system, there are currently a limited number of features that researchers can use to detect malware by using machine learning methods. The analysis and summary we presented in subsection IV-C indicates that most of the research is still based on common features such as permission and API calls. In our opinion, some future directions for feature extraction and establishment are as follows. Firstly, we can search for the relationship between different types of information about Android applications to establish features. Secondly, we can combine these commonly used features across the time and space domains to establish new features. Thirdly, we can also use third-party platforms or crowdsourcing to obtain auxiliary feature data for malware analysis and detection.

D. APPLICATION OF MACHINE LEARNING

Machine learning has rapidly growing in recent years. By referring to the latest research achievements in machine learning and artificial intelligence, machine learning methods should be carried out with the goal of improving the effectiveness and efficiency of detection.

The design of the algorithm should be more inclined toward mixed and multi-level techniques. The subject of Android malware detection can be assessed from different perspectives such as image processing, natural language processing, and data mining. Combined with deep learning, online learning, and other methods, hybrid feature data obtained from multiple channels can be used to achieve a more intelligent and adaptive detection effect. To the best of our knowledge, many learning algorithms that have been paid attention to and applied in other fields have not been widely used in Android malware detection [24]. For example, incremental learning can dynamically add sample data to maintain the high performance of the classifier [289], active learning can deal with the data scarcity problem and reduce the learning cost [290], and transfer learning can apply the knowledge obtained from learning tasks to improve the learning effect on other related tasks [291]. From a macro perspective, the problems that can be solved by these learning algorithms are present in the field of Android malware detection. In the future, these types of learning algorithms will be applied more widely to Android malware detection.

Another research direction that cannot be ignored in machine learning is concept drift [292]. Over time, due to the continuous development and evolution of Android malware, the predictive performance of a trained classifier declines. This phenomenon is called concept drift, which is a problem that researchers have been trying to solve in the field of machine learning. Although the problem of concept drift can be alleviated by using new datasets for training periodically to update the classifier's models, it is obvious that the overhead is large, and the classifier's prediction effectiveness between two training cycles cannot be guaranteed. The Transcend framework [293] utilizes statistical methods to detect concept drift and is not subject to the machine learning algorithm used by the classifier; however, it does not propose a specific and effective method for the classifier to overcome concept drift. DroidSpan [294] defines two parameters to evaluate the stability of the classifier under the influence of factors such as concept drift, and further makes use of a set of dynamic features with stable and distinct differentiation to train the classifier. Although the predictions are more stable than the existing tools or algorithms, the prediction accuracy is still greatly affected by concept drift and other factors, and may be thwarted by code obfuscation, the addition of glue code, time bombs, and other malicious countermeasures. To overcome the problem of classifier aging, DroidEvolver [288] obtains API call features through static analysis and uses the ensemble method combined with online learning to update the models integrated in the classifier. Although this method is somewhat robust to code obfuscation, static analysis is affected by techniques such as dynamic code loading, and online learning is threatened by sabotage methods such as poisoning attacks [295].

The robustness of detection methods based on machine learning has received more and more attention from researchers. Machine learning techniques were not originally designed to deal with purposeful and capable attackers. Some studies have shown that machine learning has inherent weaknesses, and attackers can modify their behavior to mislead learning algorithms and thus avoid detection during testing [296], [297]. When we choose a machine learning algorithm to detect Android malware, prediction accuracy, computational complexity, etc., are often the primary factors

considered, and it is easy to ignore the security issues that machine learning algorithms may face, such as susceptibility to poisoning attacks and evasion attacks [298]. How to use adversarial learning [299] and other methods to train the learner to deal with attacks effectively, and how to strengthen the security of a machine learning algorithm without significantly increasing its cost, are some topics that need further study.

E. CLASSIFIER EVALUATION

The Android application classifier obtained by machine learning needs to be properly evaluated; otherwise, the results predicted by the classifier in the real world will be meaningless. The selection of evaluation methods and metrics is a very important topic, and should be based on sound reasoning [300]. Blindly choosing an evaluation metric and method may lead to the wrong conclusions.

As we mentioned in subsection IV-E, when researchers apply machine learning methods to the detection of Android malware, they tend to directly select specific metrics such as accuracy and precision to evaluate the classifier, without giving sufficient justification for their choice. No single evaluation method is better than the others in all cases, but some methods are superior to others under certain conditions or are obviously inadequate in particular cases. Therefore, it is better to reconsider the selection of the classifier evaluation method for every new training and testing cycle [301]. For Android malware detection, we can try to use methods such as meta learning to summarize the mapping relationship between a specific scenario and algorithm performance [302]–[304], thereby helping us identify appropriate evaluation methods and metrics. On this basis, the development of a unified automated evaluation framework will help to improve the efficiency of classifier evaluation. We can further expand the evaluation metrics for Android application classifiers, such as robustness, confidence, and generalization capability. At present, most learning algorithms have a high computational cost, which limits their applicability in many practical scenarios. Therefore, the real-time performance of the classifier deserves further study. On the basis of the existing evaluation metrics, we can construct new evaluation metrics that are multi-level or multi-dimensional. For example, the work in Ref. [258] shows that a two-tier evaluation combining the metrics of AUC and accuracy has a better performance than an evaluation of AUC or accuracy alone. In addition, the design of evaluation methods with lower computational cost and higher efficiency, systematic evaluation of the correlation between different evaluation metrics, and exploration of the relationship between evaluation methods and test adequacy are all areas that invite further study.

For the time being, the reliability estimation for the evaluation results of classifiers has not been widely studied in the field of Android malware detection. Many theories and techniques in statistics, such as hypothesis testing or confidence measures [305], can be applied in this research. There are more studies on the reliability estimation of the results in traditional statistics and other application fields [306]–[308], but it does not seem to have attracted enough attention in the field of Android malware detection. This phenomenon can be attributed to the fact that most researchers focus their evaluation on the performance of classifiers, and the fact that many studies are based on an established and labeled dataset, without considering the use of classifiers in real-world environments [281]. As we mentioned in subsection IV-E, hypothesis testing can be simply regarded as a choice between two opposing hypotheses (the null hypothesis and the alternative hypothesis). A hypothesis test uses limited observational data in order to either reject or accept the null hypothesis [309], [310]. A hypothesis test does not prove a hypothesis, but merely provides evidence to accept or reject it. The p-value, representing probability, measures the strength of evidence against the null hypothesis [311]. It is affected by sample size, sample distribution, and other factors. After obtaining the p-value through hypothesis testing, it is necessary to carry out targeted explanation of the p-value in specific application scenarios, which will make the hypothesis testing more meaningful. In addition, the calculated accurate p-value and the relevant confidence interval can be given together to provide a more comprehensive evaluation of the evaluation results. Diversity, robustness, and applicability are also factors to be considered [312]. The research on reliability estimation can be used to evaluate the application effect of selected algorithms and features, which can in turn improve their selection. We believe that the study of classifier evaluation can fully draw on and utilize the latest research theories and results of statistical analysis.

VI. CONCLUSION

With the popularization of the Internet of Things, 5G, and other technologies, mobile smart devices are developing rapidly, and the scale of Android applications installed on smart terminals, such as mobile phones and tablets, is also increasing. However, this has been followed by an increase in malware targeting the platform. In turn, this has attracted a great deal of research into detecting Android applications that are affected by malware. The introduction of artificial intelligence methods, such as machine learning, has greatly improved the prospects for the detection of Android malware. Through surveying the collected literature, this paper provides a detailed review of current approaches for detecting Android malware, with a focus on the use of machine learning. The main aim of this paper is to present a complete picture of Android malware detection based on machine learning.

We briefly introduced the background to Android malware and gave a comprehensive review of machine learning-based approaches for detecting Android malware, arranged roughly in the order of the machine learning development pipeline. A range of alternative approaches were considered at each stage with a detailed consideration of their advantages in specific contexts. Some key content was summarized in the form of diagrams and tables to provide a better understanding

and facilitate comparative analysis. Finally, we isolated five topics to be studied in future research: the establishment of the sample set, data optimization and processing, feature extraction and establishment, application of machine learning, and classifier evaluation.

This work is different from previous surveys on Android malware detection, focusing on more aspects of machine learning methods. We believe this work complements previous reviews by filling some research gaps and putting forward some open issues in this field. We hope this review will provide a foundation for interested readers and inspire them to pursue new research avenues.

REFERENCES

[1] *Smartphone Market Share*. Accessed: Apr. 30, 2020. [Online]. Available: https://www.idc.com/promo/smartphone-market-share/os

[2] *Number of Android Apps on Google Play*. Accessed: Apr. 30, 2020. [Online]. Available: https://www.appbrain.com/stats/number-of-android-apps

[3] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, "Significant permission identification for Machine-Learning-Based Android malware detection," *IEEE Trans. Ind. Informat.*, vol. 14, no. 7, pp. 3216–3225, Jul. 2018.

[4] Sufatrio, D. J. J. Tan, T. W. Chua, and V. L. L. Thing, "Securing Android: A survey, taxonomy, and challenges," *ACM Comput. Surv.*, vol. 47, no. 4, p. 58, May 2015.

[5] S. H. Qing, "Research progress on Android security," *J. Softw.*, vol. 27, no. 1, pp. 45–71, Jan. 2016.

[6] J. Lopes, C. Serrao, L. Nunes, A. Almeida, and J. Oliveira, "Overview of machine learning methods for Android malware identification," in *Proc. 7th Int. Symp. Digit. Forensics Secur. (ISDFS)*, Barcelos, Portugal, Jun. 2019, pp. 1–6.

[7] M. Choudhary and B. Kishore, "HAAMD: Hybrid analysis for Android malware detection," in *Proc. Int. Conf. Comput. Commun. Informat. (ICCCI)*, Jan. 2018, pp. 1–4.

[8] M. Taleby, Q. Li, M. Rabbani, and A. Raza, "A survey on smartphones security: Software vulnerabilities, malware, and attacks," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 10, pp. 30–45, 2017.

[9] A. Souri and R. Hosseini, "A state-of-the-art survey of malware detection approaches using data mining techniques," *Hum.-Centric Comput. Inf. Sci.*, vol. 8, no. 1, p. 22, Jan. 2018.

[10] A. Amamra, C. Talhi, and J.-M. Robert, "Smartphone malware detection: From a survey towards taxonomy," in *Proc. 7th Int. Conf. Malicious Unwanted Softw.*, Fajardo, PR, USA, Oct. 2012, pp. 79–86.

[11] H. Lubuva, Q. Huang, and G. C. Msonde, "A review of static malware detection for Android apps permission based on deep learning," *Int. J. Comput. Netw. Appl.*, vol. 6, no. 5, pp. 80–91, Sep./Oct. 2019.

[12] E. J. Alqahtani, R. Zagrouba, and A. Almuhaideb, "A survey on Android malware detection techniques using machine learning algorithms," in *Proc. 6th Int. Conf. Softw. Defined Syst. (SDS)*, Rome, Italy, Jun. 2019, pp. 110–117.

[13] A. Naway and Y. LI, "A review on the use of deep learning in Android malware detection," 2018, *arXiv:1812.10360*. [Online]. Available: http://arxiv.org/abs/1812.10360

[14] S. Hahn, M. Protsenko, and T. Müller, "Comparative evaluation of machine learning-based malware detection on Android," in *Lecture Notes in Informatics Sicherheit 2016-Sicherheit, Schutz und Zuverlässigkeit*, M. Meier, D. Reinhardt, and S. Wendzel, Eds. Bonn, Germany: Gesellschaft für Informatik, 2016, pp. 77–88.

[15] V. Kouliaridis, K. Barmpatsalou, G. Kambourakis, and S. Chen, "A survey on mobile malware detection techniques," *IEICE Trans. Inf. Syst.*, vol.103, no. 2, pp. 204–211, Feb. 2020.

[16] A. A. A. Samra, H. N. Qunoo, F. Al-Rubaie, and H. El-Talli, "A survey of static Android malware detection techniques," in *Proc. IEEE 7th Palestinian Int. Conf. Electr. Comput. Eng. (PICECE)*, Mar. 2019, pp. 1–6.

[17] A. Qamar, A. Karim, and V. Chang, "Mobile malware attacks: Review, taxonomy & future directions," *Future Gener. Comput. Syst.*, vol. 97, pp. 887–909, Aug. 2019.

[18] Y. S. I. Hamed, S. N. A. AbdulKader, and M. M. Mostafa, "Mobile malware detection: A survey," *Int. J. Comput. Sci. Inf. Secur.*, vol. 17, no. 1, pp. 56–65, Jan. 2019.

[19] P. Yan and Z. Yan, "A survey on dynamic mobile malware detection," *Softw. Qual. J.*, vol. 26, no. 3, pp. 891–919, Sep. 2018.

[20] M. Odusami, O. Abayomi-Alli, S. Misra, O. Shobayo, R. Damasevicius, and R. Maskeliunas, "Android malware detection: A survey," in *Proc. Int. Conf. Appl. Inform. (ICAI)*, Bogota, Colombia, 2018, pp. 255–266.

[21] D. BalaGanesh, A. Chakrabarti, and D. Midhunchakkaravarthy, "Smart devices threats, vulnerabilities and malware detection approaches: A survey," *Eur. J. Eng. Res. Sci.*, vol. 3, no. 2, pp. 7–12, Feb. 2018.

[22] N. K. Gyamfi and E. Owusu, "Survey of mobile malware analysis, detection techniques and tool," in *Proc. IEEE 9th Annu. Inf. Technol., Electron. Mobile Commun. Conf. (IEMCON)*, Vancouver, BC, Canada, Nov. 2018, pp. 1101–1107.

[23] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, "The evolution of Android malware and Android analysis techniques," *ACM Comput. Surv.*, vol. 49, no. 4, p. 76, 2017.

[24] Y. Ye, T. Li, D. A. Adjeroh, and S. S. Iyengar, "A survey on malware detection using data mining techniques," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 41, 2017, doi: 10.1145/3073559.

[25] S. K. Muttoo and S. Badhani, "Android malware detection: State of the art," *Int. J. Inf. Technol.*, vol. 9, no. 1, pp. 111–117, Mar. 2017.

[26] L. Aneja and S. Babbar, "Research trends in malware detection on Android devices," in *Proc. Int. Conf. Recent Develop. Sci., Eng. Technol.*, Gurgaon, India, 2017, pp. 629–642.

[27] S. Arshad, M. A. Shah, A. Khan, and M. Ahmed, "Android malware detection & protection: A survey," *Int. J. Adv. Comput. Sci. Appl.*, vol. 7, no. 2, pp. 463–475, Feb. 2016.

[28] N. Yadav, A. Sharma, and A. Doegar, "A survey on Android malware detection," *Int. J. New Technol. Res.*, vol. 2, no. 12, pp. 47–53, Dec. 2016.

[29] M. Sharma, M. Chawla, and J. Gajrani, "A survey of Android malware detection strategy and techniques," in *Proc. Int. Conf. ICT Sustain. Develop.*, Ahmedabad, India, 2015, pp. 39–51.

[30] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," *Digit. Invest.*, vol. 13, pp. 22–37, Jun. 2015.

[31] A. Skovoroda and D. Gamayunov, "Review of the mobile malware detection approaches," in *Proc. 23rd Euromicro Int. Conf. Parallel, Distrib., Netw.-Based Process.*, Mar. 2015, pp. 600–603.

[32] M. Y. Ricky and R. S. Gulo, "A comprehensive study for mobile Android malware detection," in *Proc. Int. Conf. Netw. Secur. Comput. Sci. (ICNSCS)*, Antalya, Turkey, 2015, pp. 13–17.

[33] L. Dua and D. Bansal, "Review on mobile threats and detection techniques," *Int. J. Distrib. Parallel Syst.*, vol. 5, no. 4, pp. 21–29, Jul. 2014.

[34] L. Dua and D. Bansal, "Taxonomy: Mobile malware Threats and detection techniques," in *Proc. 4th Int. Conf. Adv. Comput. Inf. Technol.*, Delhi, India, 2014, pp. 213–221.

[35] P. Bhat and K. Dutta, "A survey on various threats and current state of security in Android platform," *ACM Comput. Surv.*, vol. 52, no. 1, p. 21, Feb. 2019.

[36] L. Li, T. F. Bissyandé, M. Papadakis, S. Rasthofer, A. Bartel, D. Octeau, J. Klein, and L. Traon, "Static analysis of Android apps: A systematic literature review," *Inf. Softw. Technol.*, vol. 88, pp. 67–95, Aug. 2017.

[37] M. Xu, C. Song, Y. Ji, M.-W. Shih, K. Lu, C. Zheng, and R. Duan, "Toward engineering a secure Android ecosystem: A survey of existing techniques," *ACM Comput. Surv.*, vol. 49, no. 2, p. 38, 2016.

[38] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, "Android security: A survey of issues, malware penetration, and defenses," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 2, pp. 998–1022, 2nd Quart., 2015.

[39] M. L. Polla, F. Martinelli, and D. Sgandurra, "A survey on security for mobile devices," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 1, pp. 446–471, 1st Quart., 2012.

[40] *ACM Digital Libraries*. Accessed: Apr. 30, 2020. [Online]. Available: https://dl.acm.org

[41] *Science Direct*. Accessed: Apr. 30, 2020. [Online]. Available: https://www.sciencedirect.com

[42] *Web of Science*. Accessed: Apr. 30, 2020. [Online]. Available: https://webofknowledge.com

[43] *IEEE Xplore Digital Library*. Accessed: Apr. 30, 2020. [Online]. Available: https://ieeexplore.ieee.org

[44] *Cornell University Library*. Accessed: Apr. 30, 2020. [Online]. Available: https://arxiv.org

[45] *SpringerLink*. Accessed: Apr. 30, 2020. [Online]. Available: https://link.springer.com

[46] *GoogleScholar*. Accessed: Apr. 30, 2020. [Online]. Available: https://scholar.google.com/
[47] *ResearchGate*. Accessed: Apr. 30, 2020. [Online]. Available: https://www.researchgate.net/
[48] *Academia*. Accessed: Apr. 30, 2020. [Online]. Available: https://www.academia.edu/
[49] S. Jalali and C. Wohlin, "Systematic literature studies: Database searches vs. backward snowballing," in *Proc. ACM-IEEE Int. Symp. Empirical Softw. Eng. Meas.*, Lund, Sweden, Sep. 2012, pp. 29–38.
[50] *Secure an Android Device*. Accessed: Apr. 30, 2020. [Online]. Available: https://source.android.com/security
[51] *Platform Architecture*. Accessed: Apr. 30, 2020. [Online]. Available: https://developer.android.com/guide/platform
[52] X. Q. Wang, Y. W. Wang, L. M. Liu, L. G. Lei, and J. W. Jing, "WrapDroid: Flexible and fine-grained scheme towards regulating behaviors of Android apps," in *Proc. Int. Conf. Inf. Secur. Cryptol. (ICISC)*, Seoul, South Korea, 2014, pp. 255–268.
[53] N. Elenkov, *Android Security Internals: An In-depth Guide to Android's Security Architecture*. San Francisco, CA, USA: No Starch Press, 2015.
[54] *Android 10*. Accessed: Apr. 30, 2020. [Online]. Available: https://developer.android.google.cn/about/versions/10
[55] E. Skoudis, and L. Zeltser, *Malware: Fighting Malicious Code*. Upper Saddle River, NJ, USA: Prentice-Hall, 2004.
[56] M. Sikorski, and A. Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. San Francisco, CA, USA: No Starch Press, 2012.
[57] Y. Zhang, K. Wang, H. Yang, Z. Fang, Z. Wang, and C. Cao, "Survey of Android OS security," *J. Comput. Res. Develop.*, vol. 51, no. 7, pp. 1385–1396, Jul. 2014.
[58] J. Liu, P. R. Su, M. Yang, L. He, Y. Zhang, X. Y. Zhu, and H. M. Lin, "Software and cyber security-A survey," *J. Softw.*, vol. 29, no. 1, pp. 42–68, Jan. 2018.
[59] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, detection and analysis of malware for smart devices," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 961–987, 2nd Quart., 2014.
[60] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proc. 1st ACM Workshop Secur. Privacy Smartphones Mobile Devices (SPSM)*, Chicago, IL, USA, 2011, pp. 3–14.
[61] Y. Zhou and X. Jiang, "Dissecting Android malware: Characterization and evolution," in *Proc. IEEE Symp. Secur. Privacy*, San Francisco, CA, USA, May 2012, pp. 95–109.
[62] *Android Security & Privacy 2018 Year In Review*. Accessed: Apr. 30, 2020. [Online]. Available: https://source.android.com/security/reports/Google_Android_Security_2018_Report_Final.pdf
[63] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine Learning: An Artificial Intelligence Approach*. San Mateo, CA, USA: Morgan Kaufmann, 1983.
[64] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997.
[65] R. E. Schapire, "The boosting approach to machine learning: An overview," in *Nonlinear Estimation and Classification*. New York, NY, USA: Springer, 2003.
[66] P. Domingos, *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. New York, NY, USA: Basic Books, 2015.
[67] S. B. Kotsiantis, I. D. Zaharakis, and P. E. Pintelas, "Machine learning: A review of classification and combining techniques," *Artif. Intell. Rev.*, vol. 26, no. 3, pp. 159–190, Nov. 2006.
[68] X. R. Li and X. H. Ding, "Survey on five tribes of machine learning and the main algorithms," *Softw. Guide*, vol. 18, no. 7, pp. 4–9, Jul. 2019.
[69] A. Dey, "Machine learning algorithms: A review," *Int. J. Comput. Sci. Inf. Technol.*, vol. 7, no. 3, pp. 1174–1179, 2016.
[70] *Machine Learning*. Accessed: Apr. 30, 2020. [Online]. Available: https://en.wikipedia.org/wiki/Machine_learning
[71] M. Mohammed, M. B. Khan, and E. B. M. Bashier, *Machine Learning: Algorithms and Applications*. Boca Raton, FL, USA: CRC Press, 2016.
[72] K. Das and R. N. Behera, "A survey on machine learning: Concept, algorithms and applications," *Int. J. Innov. Res. Comput. Commun. Eng.*, vol. 5, no. 2, pp. 1301–1309, Feb. 2017.
[73] V. J. Prakash and L. M. Nithya, "A survey on semi-supervised learning techniques," *Int. J. Comput. Trends Technol.*, vol. 8, no. 1, pp. 25–29, Feb. 2014.
[74] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *J. Artif. Intell. Res.*, vol. 4, no. 1, pp. 237–285, Jan. 1996.
[75] P. Langley and H. A. Simon, "Applications of machine learning and rule induction," *Commun. ACM*, vol. 38, no. 11, pp. 54–64, Nov. 1995.
[76] M. Mohri, A. Rostamizaden, and A. Talwalkar, *Foundations of Machine Learning*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.
[77] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: MIT Press, 2012.
[78] S. Marsland, *Machine Learning: An Algorithmic Perspective*, 2nd ed. Boca Raton, FL, USA: CRC press, 2015.
[79] Z. H. Zhou, *Machine Learning*. Beijing, China: Tsinghua Univ. Press, 2016.
[80] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep ground truth analysis of current Android malware," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*, Bonn, Germany, 2017, pp. 252–276.
[81] N. d'Heureuse, F. Huici, M. Arumaithurai, M. Ahmed, K. Papagiannaki, and S. Niccolini, "What's app? A wide-scale measurement study of smart phone markets," *Mobile Comput. Commun. Rev.*, vol. 16, no. 2, pp. 16–27, Apr. 2012.
[82] *VirusTotal*. Accessed: Apr. 30, 2020. [Online]. Available: https://www.virustotal.com
[83] *AndroBugs*. Accessed: Apr. 30, 2020. [Online]. Available: https://www.androbugs.com
[84] Y. Ishii, T. Watanabe, F. Kanei, Y. Takata, E. Shioji, M. Akiyama, T. Yagi, B. Sun, and T. Mori, "Understanding the security management of global third-party Android marketplaces," in *Proc. 2nd ACM SIGSOFT Int. Workshop App Market Anal. (WAMA)*, 2017, pp. 12–18.
[85] (2020). *VirusShare.com-Because Sharing is Caring, VirusShare*. Accessed: Apr. 30, 2020. [Online]. Available: http://virusshare.com/.
[86] (2020). *Contagio mobile, Mobile Malware Mini Dump*. Accessed: Apr. 30, 2020. [Online]. Available: http://contagiominidump.blogspot.com
[87] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of Android malware in your pocket," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 23–26.
[88] *AndroZoo*. Accessed: Apr. 30, 2020. [Online]. Available: https://androzoo.uni.lu/
[89] H. Zhang, S. Luo, Y. Zhang, and L. Pan, "An efficient Android malware detection system based on method-level behavioral semantic analysis," *IEEE Access*, vol. 7, pp. 69246–69256, 2019.
[90] Y. Li, J. Jang, X. Hu, and X. Ou, "Android malware clustering through malicious payload mining," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses (RAID)*, Atlanta, GA, USA, vol. 2017, pp. 192–214.
[91] M. Murtaz, H. Azwar, S. B. Ali, and S. Rehman, "A framework for Android Malware detection and classification," in *Proc. IEEE 5th Int. Conf. Eng. Technol. Appl. Sci. (ICETAS)*, Nov. 2018, pp. 1–5.
[92] H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Droidkin: Lightweight detection of Android apps similarity," in *Proc. Int. Conf. Secur. Privacy Commun. Netw.*, Beijing, China, 2014, pp. 436–453.
[93] J. Li, Z. Wang, T. Wang, J. Tang, Y. Yang, and Y. Zhou, "An Android malware detection system based on feature fusion," *Chin. J. Electron.*, vol. 27, no. 6, pp. 1206–1213, Nov. 2018.
[94] H. Wang, J. Si, H. Li, and Y. Guo, "RmvDroid: Towards a reliable Android malware dataset with app metadata," in *Proc. IEEE/ACM 16th Int. Conf. Mining Softw. Repositories (MSR)*, May 2019, pp. 404–408.
[95] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Comput. Electr. Eng.*, vol. 40, no. 1, pp. 16–28, Jan. 2014.
[96] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. San Mateo, CA, USA: Morgan Kaufmann, 2012.
[97] S. Garcia, J. Luengo, and F. Herrera, *Data Preprocessing in Data Mining*. New York, NY, USA: Springer, 2015.
[98] K. Kobkul, C. Gareth, and M. Phayung, "A hybrid system based on multi-agent system in the data preprocessing stage," *Int. J. Comput. Sci. Inf. Secur.*, vol. 7, no. 2, pp. 199–202, Feb. 2010.
[99] J. Sahs and L. Khan, "A machine learning approach to Android malware detection," in *Proc. Eur. Intell. Secur. Informat. Conf.*, Aug. 2012, pp. 141–147.
[100] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proc. 36th Int. Conf. Softw. Eng. (ICSE)*, 2014, pp. 1025–1035.
[101] L. Singh and M. Hofmann, "Dynamic behavior analysis of Android applications for malware detection," in *Proc. Int. Conf. Intell. Commun. Comput. Techn. (ICCT)*, Dec. 2017, pp. 1–7.
[102] S. Lou, S. Cheng, J. Huang, and F. Jiang, "TFDroid: Android malware detection by topics and sensitive data flows using machine learning techniques," in *Proc. IEEE 2nd Int. Conf. Inf. Comput. Technol. (ICICT)*, Kahului, HI, USA, Mar. 2019, pp. 30–36.

[103] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: Behavior-based malware detection system for Android," in *Proc. 1st ACM Workshop Secur. Privacy Smartphones Mobile Devices (SPSM)*, Chicago, IL, USA, 2011, pp. 15–26.

[104] M. Protsenko and T. Muller, "Android malware detection based on software complexity metrics," in *Proc. Int. Conf. Trust, Privacy Secur. Digit. Bus.*, Munich, Germany, 2014, pp. 24–35.

[105] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, "A combination method for Android malware detection based on control flow graphs and machine learning algorithms," *IEEE Access*, vol. 7, pp. 21235–21245, 2019.

[106] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "MARVIN: Efficient and comprehensive mobile app classification through static and dynamic analysis," in *Proc. IEEE 39th Annu. Comput. Softw. Appl. Conf.*, Jul. 2015, pp. 422–433.

[107] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "MaMaDroid: Detecting Android malware by building Markov chains of behavioral models," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2017, pp. 1–34.

[108] T. Chen, Q. Mao, Y. Yang, M. Lv, and J. Zhu, "TinyDroid: A lightweight and efficient model for Android malware detection and classification," *Mobile Inf. Syst.*, vol. 2018, Oct. 2018, Art. no. 4157156.

[109] J. Zhang, Z. Qin, H. Yin, L. Ou, and Y. Hu, "IRMD: Malware variant detection using opcode image recognition," in *Proc. IEEE 22nd Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2016, pp. 1175–1180.

[110] M. Yang and Q. Wen, "Detecting Android malware by applying classification techniques on images patterns," in *Proc. IEEE 2nd Int. Conf. Cloud Comput. Big Data Anal. (ICCCBDA)*, Apr. 2017, pp. 344–347.

[111] J. Yan, Y. Qi, and Q. Rao, "Detecting malware with an ensemble method based on deep neural network," *Secur. Commun. Netw.*, vol. 2018, Mar. 2018, Art. no. 7247095.

[112] W. C. Wu and S. H. Hung, "DroidDolphin: A dynamic Android malware detection framework using big data and machine learning," in *Proc. Conf. Res. Adapt. Convergent Syst.*, Towson, MD, USA, 2014, pp. 247–252.

[113] G. Canfora, A. D. Lorenzo, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Effectiveness of opcode ngrams for detection of multi family Android malware," in *Proc. 10th Int. Conf. Availability, Rel. Secur.*, Aug. 2015, pp. 333–340.

[114] B. Kang, S. Y. Yerima, K. Mclaughlin, and S. Sezer, "N-opcode analysis for Android malware classification and categorization," in *Proc. Int. Conf. Cyber Secur. Protection Digit. Services (Cyber Secur.)*, Jun. 2016, pp. 1–7.

[115] F. Martinelli, F. Mercaldo, and A. Saracino, "BRIDEMAID: An hybrid tool for accurate detection of Android malware," in *Proc. ACM Asia Conf. Comput. Commun. Secur. (CCS)*, Abu Dhabi, UAE, 2017, pp. 899–901.

[116] F. Shen, J. D. Vecchio, A. Mohaisen, S. Y. Ko, and L. Ziarek, "Android malware detection using complex-flows," *IEEE Trans. Mobile Comput.*, vol. 18, no. 6, pp. 1231–1245, Jun. 2019.

[117] A. Shabtai, Y. Fledel, Y. Elovici, and Y. Shahar, "Using the KBTA method for inferring computer and network security alerts from time-stamped, raw system metrics," *J. Comput. Virol.*, vol. 6, no. 3, pp. 239–259, Aug. 2010.

[118] X. Xiao, X. Xiao, Y. Jiang, X. Liu, and R. Ye, "Identifying Android malware with system call co-occurrence matrices," *Trans. Emerg. Telecommun. Technol.*, vol. 27, no. 5, pp. 675–684, Feb. 2016.

[119] L. Zhao, D. Li, G. Zheng, and W. Shi, "Deep neural network based on Android mobile malware detection system using opcode sequences," in *Proc. IEEE 18th Int. Conf. Commun. Technol. (ICCT)*, Oct. 2018, pp. 1141–1147.

[120] T. Gao, W. Peng, D. Sisodia, T. K. Saha, F. Li, and M. Al Hasan, "Android malware detection via graphlet sampling," *IEEE Trans. Mobile Comput.*, vol. 18, no. 12, pp. 2754–2767, Dec. 2019.

[121] M. Dash and H. Liu, "Feature selection for classification," *Intell. Data Anal.*, vol. 1, nos. 1–4, pp. 131–156, 1997.

[122] H. W. Liu, "A study on feature selection algorithms using information entropy," Ph.D. dissertation, College Comput. Sci. Technol., Jilin Univ., Jilin, China, 2010.

[123] K. Kira and L. A. Rendell, "A practical approach to feature selection," in *Proc. 9th Int. Workshop Mach. Learn.*, Aberdeen, U.K., 1992, pp. 249–256.

[124] I, Kononenko, "Estimating attributes: Analysis and extensions of RELIEF," in *Proc. Eur. Conf. Mach. Learn.*, Catania, Italy, 1994, pp. 171–182.

[125] A. K. Jain, R. P. W. Duin, and J. C. Mao, "Statistical pattern recognition: A review," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 1, pp. 4–37, Jan. 2000.

[126] J. Sheinvald, B. Dom, and W. Niblack, "A modeling approach to feature selection," in *Proc. 10th Int. Conf. Pattern Recognit.*, Atlantic, NJ, USA, Jun. 1990, pp. 535–539.

[127] G. M. Liu, "Research on two-stage feature selection methods in machine learning," M.S. thesis, School Comput. Sci. Technol., Harbin Univ. Sci. Technol., Heilongjiang, China, 2015.

[128] A. N. Mucciardi and E. E. Gose, "A comparison of seven techniques for choosing subsets of pattern recognition properties," *IEEE Trans. Comput.*, vol. C-20, no. 9, pp. 1023–1031, Sep. 1971.

[129] M. Modrzejewski, "Feature selection using rough sets theory," in *Proc. Eur. Conf. Mach. Learn.*, Vienna, Austria, 1993, pp. 213–226.

[130] X. Sun, "Research on feature selection for machine learning," Ph.D. dissertation, College Comput. Sci. Technol., Jilin Univ., Jilin, China, 2013.

[131] H. Almuallim and T. G. Dietterich, "Learning with many irrelevant features," in *Proc. 9th Nat. Conf. Artif. Intell.*, Anaheim, CA, USA, 1991, pp. 547–552.

[132] H. Liu and R. Setiono, "A probabilistic approach to feature selection-a filter solution," in *Proc. 13th Int. Conf. Mach. Learn. (ICML)*, Bari, Italy, 1996, pp. 319–327.

[133] P. A. Devijver, and J. Kittler, *Pattern Recognition: A Statistical Approach*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1982.

[134] H. Liu and R. Setiono, "Feature selection and classification-a probabilistic wrapper approach," in *9th Int. Conf. Ind. Eng. Appl. Artif. Intell. Expert Syst. (IEA/AIE)*, Fukuoka, Japan, 1996, pp. 419–424.

[135] I. Santos, F. Brezo, X. Ugarte-Pedrero, and P. G. Bringas, "Opcode sequences as representation of executables for data-mining-based unknown malware detection," *Inf. Sci.*, vol. 231, pp. 64–82, May 2013.

[136] Q. Jerome, K. Allix, R. State, and T. Engel, "Using opcode-sequences to detect malicious Android applications," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2014, pp. 914–919.

[137] S. Y. Yerima, I. Muttik, and S. Sezer, "High accuracy Android malware detection using ensemble learning," *IET Inf. Secur.*, vol. 9, no. 6, pp. 313–320, Nov. 2015.

[138] P. P. K. Chan and W.-K. Song, "Static detection of Android malware by using permissions and API calls," in *Proc. Int. Conf. Mach. Learn. Cybern.*, Jul. 2014, pp. 82–87.

[139] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Detecting Android malware using sequences of system calls," in *Proc. 3rd Int. Workshop Softw. Develop. Lifecycle Mobile (DeMobile)*, Bergamo, Italy, 2015, pp. 13–20.

[140] S. Abdulla and A. Altaher, "Intelligent approach for Android malware detection," *KSII Trans. Internet Inf. Syst.*, vol. 9, no. 8, pp. 2964–2983, Aug. 2015.

[141] A. Kapratwar, F. Di Troia, and M. Stamp, "Static and dynamic analysis of Android malware," in *Proc. 3rd Int. Conf. Inf. Syst. Secur. Privacy*, 2017, pp. 653–662.

[142] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni, "Android malware family classification based on resource consumption over time," in *Proc. 12th Int. Conf. Malicious Unwanted Softw. (MALWARE)*, Oct. 2017, pp. 31–38.

[143] C.-H. Liu, Z.-J. Zhang, and S.-D. Wang, "An Android malware detection approach using Bayesian inference," in *Proc. IEEE Int. Conf. Comput. Inf. Technol. (CIT)*, Dec. 2016, pp. 476–483.

[144] P. Zhang, S. Cheng, S. Lou, and F. Jiang, "A novel Android malware detection approach using operand sequences," in *Proc. 3rd Int. Conf. Secur. Smart Cities, Ind. Control Syst. Commun. (SSIC)*, Oct. 2018, pp. 1–5.

[145] C. Zhao, W. Zheng, L. Gong, M. Zhang, and C. Wang, "Quick and accurate Android malware detection based on sensitive APIs," in *Proc. IEEE Int. Conf. Smart Internet Things (SmartIoT)*, Aug. 2018, pp. 143–148.

[146] A. T. Kabakus, "What static analysis can utmost offer for Android malware detection," *Inf. Technol. Control*, vol. 48, no. 2, pp. 235–249, Feb. 2019.

[147] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "'Andromaly': A behavioral malware detection framework for Android devices," *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161–190, Jan. 2011.

[148] A. Sharma and S. K. Dash, "Mining API calls and permissions for Android malware detection," in *Proc. 13th Int. Conf. Cryptol. Netw. Secur.*, Heraklion, Greece, 2014, pp. 191–205.

[149] H.-J. Zhu, Z.-H. You, Z.-X. Zhu, W.-L. Shi, X. Chen, and L. Cheng, "DroidDet: Effective and robust detection of Android malware using static analysis along with rotation forest model," *Neurocomputing*, vol. 272, pp. 638–646, Jan. 2018.

[150] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in Android applications for malicious application detection," *IEEE Trans. Inf. Forensics Security*, vol. 9, no. 11, pp. 1869–1882, Nov. 2014.
[151] K. Xu, Y. Li, and R. H. Deng, "ICCDetector: ICC-based malware detection on Android," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 6, pp. 1252–1264, Jun. 2016.
[152] R. Goyal, A. Spognardi, N. Dragoni, and M. Argyriou, "SafeDroid: A distributed malware detection service for Android," in *Proc. IEEE 9th Int. Conf. Service-Oriented Comput. Appl. (SOCA)*, Macau, China, Nov. 2016, pp. 59–66.
[153] X. Li, J. Liu, Y. Huo, R. Zhang, and Y. Yao, "An Android malware detection method based on AndroidManifest file," in *Proc. 4th Int. Conf. Cloud Comput. Intell. Syst. (CCIS)*, Beijing, China, Aug. 2016, pp. 239–243.
[154] L. Wen and H. Yu, "An Android malware detection system based on machine learning," in *Proc. Green Energy Sustain. Develop. I*, Chongqing, China, 2017, Art. no. 020136.
[155] A. Firdaus, N. B. Anuar, A. Karim, and M. F. A. Razak, "Discovering optimal features using static analysis and a genetic search based method for Android malware detection," *Frontiers Inf. Technol. Electron. Eng.*, vol. 19, no. 6, pp. 712–736, Jun. 2018.
[156] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song, and H. Yu, "SAMADroid: A novel 3-level hybrid malware detection model for Android operating system," *IEEE Access*, vol. 6, pp. 4321–4339, 2018, doi: 10.1109/ACCESS.2018.2792941.
[157] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA, USA: Addison-Wesley, 1989.
[158] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Mach. Learn.*, vol. 3, nos. 2–3, pp. 95–99, 1988.
[159] B. Amro, "Malware detection techniques for mobile devices," *Int. J. Mobile Netw. Commun. Telematics*, vol. 7, nos. 4–6, pp. 1–10, Dec. 2017.
[160] K. Kavitha, P. Salini, and V. Ilamathy, "Exploring the malicious Android applications and reducing risk using static analysis," in *Proc. Int. Conf. Electr., Electron., Optim. Techn. (ICEEOT)*, Mar. 2016, pp. 1316–1319.
[161] B. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, "Android permissions: A perspective combining risks and benefits," in *Proc. 17th ACM Symp. Access Control Models Technol.*, New York, NJ, USA, 2012, pp. 13–22.
[162] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. C. Bringas, and G. Alvarez, "Puma: Permission usage to detect malware in Android," in *Proc. Int. Joint Conf. CISIS 12-ICEUTE 12-SOCO 12 Special Sessions*, Ostrava, Czech Republic, 2012, pp. 289–298.
[163] H. Yang, Y. Q. Zhang, Y. P. Hu, and Q. X. Liu, "Android malware detection method based on permission sequential pattern mining algorithm," *J. Commun.*, vol. 34, no. Z1, pp. 106–115, Aug. 2013.
[164] S. B. Almin and M. Chatterjee, "A novel approach to detect Android malware," in *Proc. Int. Conf. Adv. Comput. Technol. Appl. (ICACTA)*, Mumbai, India, 2015, pp. 407–417.
[165] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: Mining API-level features for robust malware detection in Android," in *Int. Conf. Secur. Privacy Commun. Syst.*, Sydney, NSW, Australia, 2013, pp. 86–103.
[166] C. I. Fan, H. W. Hsiao, C. H. Chou, and Y. F. Tseng, "Malware detection system based on API log data mining," in *Proc. 39th Int. Conf. Secur. Privacy Commun. Syst.*, Taiwan, China, 2015, pp. 255–260.
[167] S. Hou, A. Saas, Y. Ye, and L. Chen, "DroidDelver: An Android malware detection system using deep belief network based on API call blocks," in *Proc. Int. Conf. Web-Age Inf. Manage.*, Nanchang, China, 2016, pp. 54–66.
[168] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, "HinDroid: An intelligent Android malware detection system based on structured heterogeneous information network," in *Proc. ACM Conf. Knowl. Discovery Data Mining (KDD)*, Halifax, AB, Canada, 2017, pp. 1507–1515.
[169] D. Maiorca, F. Mercaldo, G. Giacinto, C. A. Visaggio, and F. Martinelli, "R-PackDroid: API package-based characterization and detection of mobile ransomware," in *Proc. Symp. Appl. Comput.*, Marrakech, Morocco, 2017, pp. 1718–1723.
[170] G. Tao, Z. Zheng, Z. Guo, and M. R. Lyu, "MalPat: Mining patterns of malicious and benign Android apps via permission-related APIs," *IEEE Trans. Rel.*, vol. 67, no. 1, pp. 355–369, Mar. 2018.
[171] N. McLaughlin, J. M. Rincon, B. Kang, S. Yerima, P. Miller, S. Sezer, Y. Safaei, E. Trickel, Z. Zhao, A. Doupe, and G. J. Ahn, "Deep Android malware detection," in *Proc. 7th ACM Conf. Data Appl. Secur. Privacy*, Scottsdale, AZ, USA, 2017, pp. 301–308.
[172] J. G. Puerta, B. Sanz, I. Santos, and P. G. Bringas, "Using Dalvik opcodes for malware detection on Android," in *Proc. Int. Conf. Hybrid Artif. Intell. Syst.*, Bilbao, Spain, 2015, pp. 416–426.
[173] A. Sharma and S. K. Sahay, "An investigation of the classifiers to detect Android malicious apps," in *Proc. 2nd Int. Congr. Inf. Commun. Technol. (ICICT)*, Bangkok, Thailand, 2016, pp. 207–217.
[174] H. Gascon, F. Yamaguchi, D. Arp, and K. Rieck, "Structural detection of Android malware using embedded call graphs," in *Proc. ACM workshop Artif. Intell. Secur. (AISec)*, 2013, pp. 45–54.
[175] A. Choliy, F. Li, and T. Gao, "Obfuscating function call topography to test structural malware detection against evasion attacks," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Jan. 2017, pp. 808–813.
[176] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, and P. Porras, "DroidMiner: Automated mining and characterization of fine-grained malicious behaviors in Android applications," in *Proc. Eur. Symp. Res. Comput. Secur. (ESORICS)*, Wroclaw, Poland, 2014, pp. 163–182.
[177] A. Martín, H. D. Menéndez, and D. Camacho, "MOCDroid: multi-objective evolutionary classifier for Android malware detection," *Soft Comput.*, vol. 21, no. 24, pp. 7405–7415, Dec. 2017.
[178] N. Peiravian and X. Zhu, "Machine learning for Android malware detection using permission and API calls," in *Proc. IEEE 25th Int. Conf. Tools With Artif. Intell.*, Nov. 2013, pp. 300–305.
[179] W. Li, J. Ge, and G. Dai, "Detecting malware for Android platform: An SVM-based approach," in *Proc. IEEE 2nd Int. Conf. Cyber Secur. Cloud Comput.*, Nov. 2015, pp. 464–469.
[180] Y. Li, Y. Ma, M. Chen, and Z. Dai, "A detecting method for malicious mobile application based on incremental SVM," in *Proc. 3rd IEEE Int. Conf. Comput. Commun. (ICCC)*, Dec. 2017, pp. 1246–1250.
[181] Y. Sun, Y. Xie, Z. Qiu, Y. Pan, J. Weng, and S. Guo, "Detecting Android malware based on extreme learning machine," in *Proc. IEEE 15th Int. Conf. Dependable, Autonomic Secure Comput., 15th Int. Conf. Pervas. Intell. Comput., 3rd Int. Conf. Big Data Intell. Comput. Cyber Sci. Technol. Congress(DASC/PiCom/DataCom/CyberSciTech)*, Nov. 2017, pp. 47–53.
[182] L. Jing, "Mobile Internet malicious application detection method based on support vector machine," in *Proc. Int. Conf. Smart Grid Electr. Autom. (ICSGEA)*, May 2017, pp. 260–263.
[183] D. Li, Z. Wang, and Y. Xue, "Fine-grained Android malware detection based on deep learning," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, May 2018, pp. 1–2.
[184] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "DroidMat: Android malware detection through manifest and API calls tracing," in *Proc. 7th Asia Joint Conf. Inf. Secur.*, Aug. 2012, pp. 62–69.
[185] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck, "MAST: Triage for market-scale mobile malware analysis," in *Proc. 6th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, Budapest, Hungary, 2013, pp. 13–24.
[186] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, J. Nieves, P. G. Bringas, and G. Álvarez Marañón, "MAMA: Manifest analysis for malware detection in Android," *Cybern. Syst.*, vol. 44, nos. 6–7, pp. 469–488, Oct. 2013.
[187] F. Idrees and M. Rajarajan, "Investigating the Android intents and permissions for malware detection," in *Proc. IEEE 10th Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2014, pp. 354–358.
[188] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "ANASTASIA: Android mAlware detection using STatic analySIs of applications," in *Proc. 8th IFIP Int. Conf. New Technol., Mobility Secur. (NTMS)*, Nov. 2016, pp. 1–5.
[189] K. Tian, D. Yao, B. G. Ryder, G. Tan, and G. Peng, "Detection of repackaged Android malware with code-heterogeneity features," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 1, pp. 64–77, Jan. 2020.
[190] D. Li, Z. Wang, and Y. Xue, "DeepDetector: Android malware detection using deep neural network," in *Proc. Int. Conf. Adv. Comput. Commun. Eng. (ICACCE)*, Jun. 2018, pp. 184–188.
[191] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu, and X. Zhang, "Droidensemble: Detecting Android malicious applications with ensemble of string and structural static features," *IEEE Access*, vol. 6, pp. 31798–31807, 2018.
[192] J. D. Koli, "RanDroid: Android malware detection using random machine learning classifiers," in *Proc. Technol. Smart-City Energy Secur. Power (ICSESP)*, Mar. 2018, pp. 1–6.
[193] J. McGiff, W. G. Hatcher, J. Nguyen, W. Yu, E. Blasch, and C. Lu, "Towards multimodal learning for Android malware detection," in *Proc. Int. Conf. Comput., Netw. Commun. (ICNC)*, Feb. 2019, pp. 432–436.

[194] U. S. Jannat, S. M. Hasnayeen, M. K. Bashar Shuhan, and M. S. Ferdous, "Analysis and detection of malware in Android applications using machine learning," in *Proc. Int. Conf. Electr., Comput. Commun. Eng. (ECCE)*, Feb. 2019, pp. 1–7.

[195] W. Enck, P. Gilbert, B. G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: An information-flow tracking system for real-time privacy monitoring on smartphones," Presented at the 9th USENIX Symp. Operating Syst. Design Implement., Vancouver, BC, Canada, Oct. 2010.

[196] V. G. Shankar, G. Somani, M. S. Gaur, V. Laxmi, and M. Conti, "Andro-Taint: An efficient Android malware detection framework using dynamic taint analysis," in *Proc. ISEA Asia Secur. Privacy (ISEASP)*, Jan. 2017, pp. 1–13.

[197] J. Zhang, Z. Qin, K. Zhang, H. Yin, and J. Zou, "Dalvik opcode graph based Android malware variants detection using global topology features," *IEEE Access*, vol. 6, pp. 51964–51974, 2018.

[198] X. Xiao, Z. Wang, Q. Li, S. Xia, and Y. Jiang, "Back-propagation neural network on Markov chains from system call sequences: A new approach for detecting Android malware with system call sequences," *IET Inf. Secur.*, vol. 11, no. 1, pp. 8–15, Jan. 2017.

[199] T. Bhatia and R. Kaushal, "Malware detection in Android based on dynamic analysis," in *Proc. Int. Conf. Cyber Secur. Protection Digit. Services (Cyber Secur.)*, Jun. 2017, pp. 1–6.

[200] S. Xu, X. Ma, Y. Liu, and Q. Sheng, "Malicious application dynamic detection in real-time API analysis," in *Proc. IEEE Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData)*, Dec. 2016, pp. 788–794.

[201] A. Feizollah, N. B. Anuar, R. Salleh, F. Amalina, R. U. R. Ma'arof, and S. Shamshirband, "A study of machine learning classifiers for anomaly-based mobile botnet detection," *Malaysian J. Comput. Sci.*, vol. 26, no. 4, pp. 251–265, Dec. 2013.

[202] A. Arora, S. Garg, and S. K. Peddoju, "Malware detection using network traffic analysis in Android based mobile devices," in *Proc. 8th Int. Conf. Next Gener. Mobile Apps, Services Technol.*, Sep. 2014, pp. 66–71.

[203] S. Garg, S. Peddoju, and A. K. Sarje, "Network-based detection of Android malicious apps," *Int. J. Inf. Secur.*, vol. 16, no. 4, pp. 385–400, 2017.

[204] M. S. Alam and S. T. Vuong, "Random forest classification for detecting Android malware," in *Proc. IEEE Int. Conf. Green Comput. Commun. IEEE Internet Things IEEE Cyber, Phys. Social Comput.*, Aug. 2013, pp. 663–669.

[205] V. M. Afonso, M. F. de Amorim, A. R. A. Grégio, G. B. Junquera, and P. L. de Geus, "Identifying Android malware using dynamically obtained features," *J. Comput. Virol. Hacking Techn.*, vol. 11, no. 1, pp. 9–17, Feb. 2015.

[206] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro, "DroidScribe: Classifying Android malware based on runtime behavior," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2016, pp. 252–261.

[207] H. Cai, N. Meng, B. Ryder, and D. Yao, "DroidCat: Effective Android malware detection and categorization via app-level profiling," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 6, pp. 1455–1470, Jun. 2019.

[208] W. Du, "Android malware detection and analysis of malware behavior," M.S. thesis, School Comput. Sci., Beijing Univ. Posts Telecommun., Beijing, China, 2018.

[209] P. D. Sawle and A. B. Gadicha, "Analysis of malware detection techniques in Android," *Int. J. Comput. Sci. Mobile Comput.*, vol. 3, no. 3, pp. 176–182, Mar. 2014.

[210] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: Deep learning in Android malware detection," in *Proc. ACM Conf. SIGCOMM*, Chicago, IL, USA, 2014, pp. 371–372.

[211] M. Spreitzenbarth, T. Schreck, F. Echtler, D. Arp, and J. Hoffmann, "Mobile-Sandbox: Combining static and dynamic analysis with machine-learning techniques," *Int. J. Inf. Secur.*, vol. 14, no. 2, pp. 141–153, Apr. 2015.

[212] S. Bhandari, R. Gupta, V. Laxmi, M. S. Gaur, A. Zemmari, and M. Anikeev, "DRACO: DRoid analyst combo an Android malware analysis framework," in *Proc. 8th Int. Conf. Secur. Inf. Netw.*, Sochi, Russia, 2015, pp. 283–289.

[213] Y. Liu, Y. Zhang, H. Li, and X. Chen, "A hybrid malware detecting scheme for mobile Android applications," in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*, Las Vegas, NV, USA, Jan. 2016, pp. 155–156.

[214] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "Stormdroid: A streaminglized machine learning-based system for detecting Android malware," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, Xi'an, China, 2016, pp. 377–388.

[215] M.-Y. Su and K.-T. Fung, "Detection of Android malware by static analysis on permissions and sensitive functions," in *Proc. 8th Int. Conf. Ubiquitous Future Netw. (ICUFN)*, Jul. 2016, pp. 873–875.

[216] S. Kandukuru and R. M. Sharma, "Android malicious application detection using permission vector and network traffic analysis," in *Proc. 2nd Int. Conf. Converg. Technol. (I2CT)*, Apr. 2017, pp. 1126–1132.

[217] L. Wei, W. Luo, J. Weng, Y. Zhong, X. Zhang, and Z. Yan, "Machine learning-based malicious application detection of Android," *IEEE Access*, vol. 5, pp. 25591–25601, 2017.

[218] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and efficient behavior-based Android malware detection and prevention," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 1, pp. 83–97, Jan. 2018.

[219] *Scikit-Learn: Machine Learning in Python*. Accessed: Apr. 30, 2020. [Online]. Available: https://scikit-learn.org

[220] H. Zhang, "The optimality of Naive Bayes," in *Proc. 7th Int. Florida Artif. Intell. Res. Soc. Conf. (FLAIRS)*, Miami Beach, FL, USA, 2004, pp. 562–567.

[221] S. F. Ding, B. J. Qi, and H. Y. Tan, "An overview on theory and algorithm of support vector machines," *J. Univ. Electron. Sci. Technol. China*, vol. 40, no. 1, pp. 2–10, Jan. 2011.

[222] P. Harrington, *Machine Learning in Action. South Hill Drive*. Westampton, NJ, USA: Manning Publications, 2012.

[223] M. H. Tian, "Research on Android malicious application detection based on machine learning," M.S. thesis, School Electron. Inf. Eng., Beijing Jiaotong Univ., Beijing, China, 2017.

[224] E. Alpaydin, *Introduction to Machine Learning*, 4th ed. Cambridge, MA, USA: MIT Press, 2020.

[225] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[226] T. Kohonen, "An introduction to neural computing," *Neural Netw.*, vol. 1, no. 1, pp. 3–16, Jan. 1988.

[227] M. Zhang, *All Illustrated Guide to Deep Learning*. Beijing, China: Posts and Telecom Press, 2018.

[228] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, Dec. 1943.

[229] D. O. Hebb, *The Organization of Behavior-A Neuropsychological Theory*. Hoboken, NJ, USA: Wiley, 1949.

[230] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning Internal Representations By Error Propagation*. Cambridge, MA, USA: MIT Press, 1988.

[231] N. Le Roux and Y. Bengio, "Representational power of restricted boltzmann machines and deep belief networks," *Neural Comput.*, vol. 20, no. 6, pp. 1631–1649, Jun. 2008.

[232] G. E. Hinton, "A practical guide to training restricted Boltzmann machines," in *Neural Networks: Tricks of the Trade*. New York, NY, USA: Springer, 2012.

[233] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, no. 8, pp. 1771–1800, Aug. 2002.

[234] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[235] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015.

[236] M. Rahman Minar and J. Naher, "Recent advances in deep learning: An overview," 2018, *arXiv:1807.08169*. [Online]. Available: http://arxiv.org/abs/1807.08169

[237] C. Zhang, and Y. Ma, *Ensemble Machine Learning: Methods and Applications*. New York, NY, USA: Springer, 2012.

[238] L. Rokach, "Ensemble-based classifiers," *Artif. Intell. Rev.*, vol. 33, nos. 1–2, pp. 1–39, 2010.

[239] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, Aug. 1996.

[240] E. Goel and E. Abhilasha, "Random forest: A review," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 7, no. 1, pp. 251–257, Jan. 2017.

[241] N. Dabbagh, B. Bannan-Ritland, *Online Learning: Concepts, Strategies, and Application*. Upper Saddle River, NJ, USA: Prentice-Hall, 2005.

[242] Y. W. Xu, *All Illustrated Guide to Machine Learning*. Beijing, China: Posts and Telecom Press, 2019.

[243] M. A. Atici, S. Sagiroglu, and I. A. Dogru, "Android malware analysis approach based on control flow graphs and machine learning algorithms," in *Proc. 4th Int. Symp. Digit. Forensic Secur. (ISDFS)*, Little Rock, AR, USA, Apr. 2016, pp. 26–31.
[244] A. Narayanan, L. Yang, L. Chen, and L. Jinliang, "Adaptive and scalable Android malware detection through online learning," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2016, pp. 2484–2491.
[245] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu, "Context-aware, adaptive, and scalable Android malware detection through online learning," *IEEE Trans. Emerg. Topics Comput. Intell.*, vol. 1, no. 3, pp. 157–175, Jun. 2017.
[246] A. K. Sikder, H. Aksu, and A. S. Uluagac, "6thSense: A context-aware sensor-based attack detector for smart devices," presented at the 26th USENIX Secur. Symp., Vancouver, BC, Canada, Aug. 2017.
[247] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: Android malware characterization and detection using deep learning," *Tsinghua Sci. Technol.*, vol. 21, no. 1, pp. 114–123, Feb. 2016.
[248] S. Hou, A. Saas, L. Chen, and Y. Ye, "Deep4MalDroid: A deep learning framework for Android malware detection based on linux kernel system call graphs," in *Proc. IEEE/WIC/ACM Int. Conf. Web Intell. Workshops (WIW)*, Omaha, NE, USA, Oct. 2016, pp. 104–111.
[249] K. Xu, Y. Li, R. H. Deng, and K. Chen, "DeepRefiner: multi-layer Android malware detection system applying deep neural networks," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Apr. 2018, pp. 473–487.
[250] A. Pektaş, M. Çavdar, and T. Acarman, "Android malware classification by applying online machine learning," in *Proc. Int. Symp. Comput. Inf. Sci.*, Krakow, Poland, 2016, pp. 72–80.
[251] B. Rashidi, C. Fung, and E. Bertino, "Android resource usage risk assessment using hidden Markov model and online learning," *Comput. Secur.*, vol. 65, pp. 90–107, Mar. 2017.
[252] M. Sokolova, N. Japkowicz, and S. Szpakowicz, "Beyond accuracy, F-score and ROC: A family of discriminant measures for performance evaluation," in *Proc. 19th Australas. Joint Conf. Artif. Intell.*, Hobart, NSW, Australia, 2006, pp. 1015–1021.
[253] P.-N. Tan, V. Kumar, and J. Srivastava, "Selecting the right objective measure for association analysis," *Inf. Syst.*, vol. 29, no. 4, pp. 293–313, Jun. 2004.
[254] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," presented at the Int. Joint Conf. Artif. Intell., Montreal, QC, Canada, Aug. 1995.
[255] N. Lavrac, P. Flach, and B. Zupan, "Rule evaluation measures: A unifying view," in *Proc. Int. Conf. Inductive Logic Program. (ILP)*, Bled, Slovenia, 1999, pp. 174–185.
[256] N. Seliya, T. M. Khoshgoftaar, and J. V. Hulse, "A study on the relationships of classifier performance metrics," in *Proc. 21st IEEE Int. Conf. Tools with Artif. Intell.*, Newark, NJ, USA, Nov. 2009, pp. 59–66.
[257] R. Caruana and A. Niculescu-Mizil, "Data mining in metric space: An empirical analysis of supervised learning performance criteria," in *Proc. 10th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Seattle, WA, USA, 2004, pp. 69–78.
[258] J. Huang and C. X. Ling, "Constructing new and better evaluation measures for machine learning," in *Proc. Int. Joint Conf. Artif. Intell.*, Hyderabad, India, 2007, pp. 859–864.
[259] E. P. Costa, A. C. Lorena, A. C. P. L. F. Carvalho, and A. A. Freitas, "A review of performance evaluation measures for hierarchical classifiers," presented at the 22nd Nat. Conf. Artif. Intell., Vancouver, BC, Canada, Jul. 2007.
[260] *Machine Learning Glossary*. Accessed: Apr. 30, 2020. [Online]. Available: https://ml-cheatsheet.readthedocs.io/en/latest#machine-learning-glossary
[261] K. A. Spackman, "Signal detection theory: Valuable tools for evaluating inductive learning," in *Proc. 6th Int. Workshop Mach. Learn.*, Ithaca, NY, USA, 1989, pp. 160–163.
[262] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognit.*, vol. 30, no. 7, pp. 1145–1159, Jul. 1997.
[263] C. X. Ling, J. Huang, and H. Zhang, "AUC: A statistically consistent and more discriminating measure than accuracy," in *18th Int. Joint Conf. Artif. Intell. (IJCAI)*, Acapulco, Mexico, 2003, pp. 519–526.
[264] D. J. Rumsey, and D. Unger, *U Can: Statistics for Dummies Cheat Sheet*. Hoboken, NJ, USA: Wiley, 2015.
[265] S. Wellek, *Testing Statistical Hypotheses of Equivalence and Noninferiority*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2010.
[266] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, 3rd ed. New York, NY, USA: McGraw-Hill, 1991.
[267] M. M. Wagner-Menghin, *Binomial Test*. Hoboken, NJ, USA: Wiley, 2014.
[268] U. Kaempf, "The binomial test: A simple tool to identify process problems," *IEEE Trans. Semiconductor Manuf.*, vol. 8, no. 2, pp. 160–166, May 1995.
[269] C. F. Gerald, and P. O. Wheatley, *Applied Numerical Analysis*, 7th ed. Reading, MA, USA: Addison-Wesley, 2004.
[270] L. Sevgi, "Hypothesis testing and decision making: Constant-False-Alarm-Rate detection," *IEEE Antennas Propag. Mag.*, vol. 51, no. 3, pp. 218–224, Jun. 2009.
[271] J. F. Box, "Guinness, Gosset, Fisher, and small samples," *Stat. Sci.*, vol. 2, no. 1, pp. 45–52, Feb. 1987.
[272] M. P. Fay and M. A. Proschan, "Wilcoxon-Mann-Whitney or t-test? On assumptions for hypothesis tests and multiple interpretations of decision rules," *Statist. Surv.*, vol. 4, no. 1, pp. 1–39, 2010.
[273] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *J. Roy. Statist. Soc. B (Methodol.)*, vol. 36, no. 2, pp. 111–147, 1974.
[274] G. Vanwinckelen and H. Blockeel, "On estimating model accuracy with repeated cross-validation," in *Proc. 21st Belgian-Dutch Conf. Mach. Learn.*, Ghent, Belgium, 2012, pp. 39–44.
[275] Q. McNemar, "Note on the sampling error of the difference between correlated proportions or percentages," *Psychometrika*, vol. 12, no. 2, pp. 153–157, Jun. 1947.
[276] P. H. Westfall, J. F. Troendle, and G. Pennello, "Multiple McNemar tests," *Biometrics*, vol. 66, no. 4, pp. 1185–1191, Dec. 2010.
[277] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *J. Amer. Stat. Assoc.*, vol. 32, no. 200, pp. 675–701, Dec. 1937.
[278] R. Eisinga, T. Heskes, B. Pelzer, and M. Te Grotenhuis, "Exact p-values for pairwise comparison of friedman rank sums, with application to comparing classifiers," *BMC Bioinf.*, vol. 18, no. 1, p. 68, Jan. 2017.
[279] P. Nemenyi, "Distribution-free multiple comparisons," Ph.D. dissertation, Dept. Math., Princeton Univ., Princeton, NJ, USA, 1963.
[280] M. Hollander, D. A. Wolfe, and E. Chicken, *Nonparametric Statistical Methods*, 3rd ed. Hoboken, NJ, USA: Wiley, 2013.
[281] K. Allix, T. F. Bissyandé, Q. Jérome, J. Klein, R. State, and Y. Le Traon, "Empirical assessment of machine learning-based malware detectors for Android," *Empirical Softw. Eng.*, vol. 21, no. 1, pp. 183–211, Feb. 2016.
[282] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in *Proc. 28th USENIX Secur. Symp.*, Santa Clara, CA, USA, 2019, pp. 729–746.
[283] M. Lindorfer, S. Volanis, A. Sisto, M. Neugschwandtner, E. Athanasopoulos, F. Maggi, C. Platzer, S. Zanero, and S. Ioannidis, "AndRadar: Fast discovery of Android applications in alternative markets," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*, Egham, U.K., vol. 2014, pp. 51–71.
[284] K. Allix, T. F. Bissyande, J. Klein, and Y. L. Traon, "Androzoo: Collecting millions of Android apps for the research community," in *Proc. IEEE/ACM 13th Working Conf. Mining Softw. Repositories (MSR)*, Austin, TX, USA, May 2016, pp. 468–471.
[285] P. Vadrevu, B. Rahbarinia, R. Perdisci, K. Li, and M. Antonakakis, "Measuring and detecting malware downloads in live network traffic," in *Proc. Eur. Symp. Res. Comput. Secur.*, Egham, U.K., 2013, pp. 556–573.
[286] V. Kumar, "Feature selection: A literature review," *Smart Comput. Rev.*, vol. 4, no. 3, pp. 211–229, Jun. 2014.
[287] P. P. Li, "Concept drifting detection and classification on data streams," Ph.D. dissertation, School Comput. Sci. Inf. Eng., Hefei Univ. Technol., Anhui, China, 2012.
[288] K. Xu, Y. Li, R. Deng, K. Chen, and J. Xu, "DroidEvolver: Self-evolving Android malware detection system," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Jun. 2019, pp. 47–62.
[289] R. R. Ade and P. R. Deshmukh, "Methods for incremental learning: A survey," *Int. J. Data Mining Knowl. Manage. Process*, vol. 3, no. 4, pp. 119–125, Jul. 2013.
[290] B. Settles, "Active learning literature survey," Univ. Wisconsin-Madison, Madison, WI, USA, Tech. Rep. 1648, Jan. 2009.
[291] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of Research on Machine Learning Applications*. Hershey, PA, USA: IGI Global, 2009, pp. 242–264.
[292] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, no. 4, p. 44, Apr. 2014. [Online]. Available: http://doi.acm.org/10.1145/2523813

[293] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *Proc. 26th USENIX Secur. Symp.*, Vancouver, BC, Canada, vol. 2017, pp. 625–642.
[294] H. Cai, "Assessing and improving malware detection sustainability through app evolution studies," *ACM Trans. Softw. Eng. Methodol.*, vol. 29, no. 2, pp. 1–28, Apr. 2020.
[295] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," in *Proc. 29th Int. Conf. Mach. Learn.*, Edinburgh, U.K., 2012, pp. 1467–1474.
[296] J. Gardiner and S. Nagaraja, "On the security of machine learning in malware C&C detection: A survey," *ACM Comput. Surv.*, vol. 49, no. 3, p. 59, Dec. 2016.
[297] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! A case study on Android malware detection," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 4, pp. 711–724, Jul. 2019.
[298] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Srndic, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*, Prague, Czech Republic, 2013, pp. 387–402.
[299] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial attacks and defences: A survey," 2018, *arXiv:1810.00069*. [Online]. Available: http://arxiv.org/abs/1810.00069
[300] N. Japkowicz, "Why question machine learning evaluation methods," in *Proc. 21st Nat. Conf. Artif. Intell. (AAAI)*, Boston, MA, USA, 2006, pp. 6–11.
[301] F. Provost and T. Fawcett, "Robust classification for imprecise environments," *Mach. Learn.*, vol. 42, no. 3, pp. 203–231, 2001.
[302] J. Vanschoren, "Meta-learning: A survey," 2018, *arXiv:1810.03548*. [Online]. Available: http://arxiv.org/abs/1810.03548
[303] C. Lemke, M. Budka, and B. Gabrys, "Metalearning: A survey of trends and technologies," *Artif. Intell. Rev.*, vol. 44, no. 1, pp. 117–130, Jun. 2015.
[304] R. Vilalta and Y. Drissi, "A perspective view and survey of meta-learning," *Artif. Intell. Rev.*, vol. 18, no. 2, pp. 77–95, 2002.
[305] M. Poggi, F. Tosi, and S. Mattoccia, "Quantitative evaluation of confidence measures in a machine learning world," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 5238–5247.
[306] S. Balakrishnan and L. Wasserman, "Hypothesis testing for high-dimensional multinomials: A selective review," 2017, *arXiv:1712.06120*. [Online]. Available: http://arxiv.org/abs/1712.06120
[307] P. Vadrevu and R. Perdisci, "Maxs: Scaling malware execution with sequential multi-hypothesis testing," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.*, Xi'an, China, 2016, pp. 771–782.
[308] S. Sohrabi, O. Udrea, and A. V. Riabov, "Hypothesis exploration for malware detection using planning," in *Proc. 27th AAAI Conf. Artif. Intell.*, Bellevue, WA, USA, 2013, pp. 883–889.
[309] A. Legay, B. Delahaye, and S. Bensalem, "Statistical model checking: An overview," in *Proc. Int. Conf. Runtime Verification*, St. Julians, Malta, 2010, pp. 122–135.
[310] C. A. Clark, "Hypothesis testing in relation to statistical methodology," *Rev. Educ. Res.*, vol. 33, no. 5, pp. 455–473, Dec. 1963.
[311] E. Whitley and J. Ball, "Statistics review 3: Hypothesis testing and *P* values," *Crit. Care*, vol. 6, no. 3, p. 222, Mar. 2002.
[312] R. S. Nickerson, "Null hypothesis significance testing: A review of an old and continuing controversy," *Psychol. Methods*, vol. 5, no. 2, pp. 241–301, Jun. 2000.

KAIJUN LIU received the M.S. degree in measuring and testing technologies and instruments from the University of Science and Technology Beijing, China, in 2013. He is currently pursuing the Ph.D. degree in cyberspace security with the Beijing University of Posts and Telecommunications, China. His research interests include mobile security and data analysis.

SHENGWEI XU received the Ph.D. degree in information security from the Graduate School, CAS, China, in 2005. He is currently an Associate Professor with the Information Security Institute, Beijing Electronic Science and Technology Institute. His research interests include big data and intelligence, and cryptographic application technology.

GUOAI XU received the Ph.D. degree in signal and information processing from the Beijing University of Posts and Telecommunications, China, in 2002. He was a Professor, in 2011. He is currently the Associate Director of the National Engineering Laboratory of Security Technology for Mobile Internet, School of Cyberspace Security, Beijing University of Posts and Telecommunications. His research interests include software security and data analysis.

MIAO ZHANG received the Ph.D. degree in signal and information processing from the Beijing University of Posts and Telecommunications, China, in 2007. He is currently an Associate Professor with the School of Cyberspace Security, Beijing University of Posts and Telecommunications. His research interests include mobile security and software security.

DAWEI SUN received the M.S. degree from the School of Information Engineering, Beijing University of Posts and Telecommunications, Beijing, China, in 2008. He is currently the Director of the Research Center for Intelligent Software Security and the Chief Technology Officer of Softsec Technologies Company Ltd. His research interests include software engineering, software security, and the mobile Internet.

HAIFENG LIU received the Ph.D. degree from the Institute of Software, Chinese Academy of Sciences, Beijing, China, in 2003. He is currently an Associate Research Fellow with the Beijing Information Security Test and Evaluation Center. His research interests include information security and data analysis.