Android Malware Detection Using API Calls: A Comparison of Feature Selection and Machine Learning Models

Android Malware Detection Using API Calls: A Comparison of Feature Selection and Machine Learning Models

Ali Muzaffar, Hani Ragab Hassen, Michael A Lones and Hind Zantout

Abstract Android has become a major target for malware attacks due its popularity and ease of distribution of applications. According to a recent study, around 11,000 new malware appear online on daily basis. Machine learning approaches have shown to perform well in detecting malware. In particular, API calls has been found to be one of the best performing features in malware detection. However, due to the functionalities provided by the Android SDK, applications can use many API calls, creating a computational overhead while training machine learning models. In this study, we look at the benefits of using feature selection to reduce this overhead. We consider three different feature selection algorithms, mutual information, variance threshold and Pearson correlation coefficient, when used with five different machine learning models: support vector machines, decision trees, random forests, Naïve Bayes and AdaBoost. We collected a dataset of 40,000 Android applications that used 134,207 different API calls. Our results show that the number of API calls can be reduced by approximately 95%, whilst still being more accurate than when the full API feature set is used. Random forests achieve the best discrimination between malware and benign applications, with an accuracy of 96.1%.

Ali Muzaffar
Heriot-Watt University, Dubai, UAE, e-mail: am29@hw.ac.uk

Hani Ragab Hassen
Heriot-Watt University, Dubai, UAE, e-mail: h.ragabhassen@hw.ac.uk

Michael A Lones
Heriot-Watt University, Edinburgh EH14 4AS, United Kingdom, e-mail: m.lones@hw.ac.uk

Hind Zantout
Heriot-Watt University, Dubai, UAE, e-mail: h.zantout@hw.ac.uk

1 Introduction

Smartphones have become an essential part of our daily lives. Today more than 48% of the world's population use smartphones, which adds up to approximately 3.8 billion people [1]. There has been an exponential growth in the number of smartphone users since 2016 when only 33.58% of the world population used smartphones [1]. Smartphone operating systems allow the users to run applications that can be downloaded from various application repositories available online. Among these operating systems, Android holds the major share of 72.18% in the smartphone market, followed by iOS at 26.96% [2].

The typical use of smartphones includes the storage of sensitive information such as text messages, emails, business data and personal files such as images and videos. Moreover, most smartphone users connect to the internet and use a variety of different services, including web and location services. Its popularity combined with the sensitive nature of the data stored in smartphones has led to an increase in the spread of Android malware.

Android allows users to download and install applications from its official application store, Google Play Store [3], and third party online stores. The existence of third party application repositories makes the spread of Android malware easier and quicker. Even Google Play Store cannot guarantee the applications listed in their store are free of malware [4].

Traditional anti-malware techniques use signature-based detection. The signature of the file can be anything from a pattern of bytes to the hash of the file. The signature is compared to a known database to detect malware [5]. However, a small modification to the signature, introduced by adding keywords or lines of code, can cause the malware to evade detection. This can prevent signature-based anti-malware from detecting existing and zero-day malware.

To overcome the limitations of signature-based detection, researchers have explored machine learning (ML) based malware detection. This process requires dataset collection, feature extraction using static and/or dynamic analysis, feature engineering and finally training ML models. Static analysis is carried out without running the application. Features are extracted by unpacking the application and mining features from the manifest and source code. One of the earlier static analysis works was by Peiravian and Zhu [6], who trained their ML models on three different feature sets including permissions (130 features), API calls (1,326 features) and a combination of both (1,456 features). They reported an accuracy of 96.88% using a support vector machine (SVM) trained on both API calls and permissions. Arp et al. [7] extracted around 545,000 features using static analysis, and reported an accuracy of 93.9%, also using an SVM. Ma et al. [8] focused on API calls, and extracted features based on API usage, frequency and sequence. Decision tree (DT), deep neural network (DNN) and long short-term memory (LSTM) models all resulted in F1 scores greater than 96%. Jung et al. [9] selected the top 50 API calls used in benign applications and malware to train a random forest model, and reported accuracies ranging from 97% to 99%.

On the other hand, features can be extracted using dynamic analysis. In this case, the application is run on an emulator and features are extracted during the runtime of the application. Afonso et al. [10] used dynamic analysis to extract system call traces and API calls used by the application. They trained a random forest model and reported an accuracy of 96.82%. Xiao et al. [11] also extracted system call traces and used them as natural language to train an LSTM language model, reporting accuracy rates of up to 96.3%. Features from both static and dynamic analysis can also be combined to build classifiers. This is called "Hybrid Analysis". For instance, Saracino et al. [12] used permissions and market information as static features and system calls, and user activity and API calls as dynamic features to train a k-nearest neighbour model. The authors reported a detection rate of 96.9%.

Dynamic analysis can be difficult to carry out because of the computational resources required to run the analysis on an Android virtual device. Therefore most researchers opt for static analysis to extract features for their ML models. Features based on API calls have produced very promising results. Therefore, we used Android API calls that were extracted using static analysis to train ML models. However, there are over 130,000 API calls that can be used by Android applications. This makes it difficult to train models on such a large feature set. To address this, we use several feature selection algorithms, namely mutual information, Pearson correlation coefficient (PCC) and variance threshold to select relevant features in Android malware detection. We then use the reduced sets of features to train SVM, random forest, Naïve bayes, DT and AdaBoost ML models. We conclude with a comparison of the results of models that were trained using different numbers of features that were selected using feature selection algorithms.

Through our comparative study we made the following contributions:

1. We compared how different ML models performed using the complete API call feature set and subsets produced by the three most commonly used feature selection algorithms in the literature.
2. We showed that random forest models perform the best with the full API feature set, reporting an accuracy rate of 95.9%. We also demonstrated that higher accuracy rates can be achieved by using only 5% of the Android API calls rather than the full API calls feature set.
3. In order to reliably evaluate these models, we collected a new, up to date, dataset of Android applications collected from various sources including popular online application stores.

The remainder of the paper is organized as follows. We introduce our framework, dataset, ML models and feature selection algorithms used in Section 2, report and discuss our findings in Section 3, followed by the conclusion in Section 4.

2 Framework Overview

The Android SDK provides programmers with API calls they can use to implement various functionalities in their applications. These functionalities include providing a GUI to the application, using hardware components of the devices and accessing user location, among many others. We crawled the Android API reference page [13] to gather all the API packages available. A Python script was then developed to extract all the API calls used by the applications by matching the API calls package name.

The aim of this study is to reduce the number of API calls used to train ML models while maintaining the detection rates produced by the full API calls feature set. The following is the framework design for the study:

- **Dataset Collection:** We collected a total of 40,000 Android applications from various sources to extract features and train classification models. These were balanced between 20,000 malware and 20,000 benign applications.
- **Feature Extraction (Static Analysis):** We wrote a Python script to extract the API calls used by the applications. *APKTool* [14] was used to reverse engineer the DEX code file to produce smali files. The smali files were then analysed to extract API calls.
- **Feature Selection:** we used Mutual information, PCC and variance threshold to select the most relevant features.
- **Train Models:** we trained ML models on the full API calls feature set and subsets produced by the feature selection methods.

2.1 Dataset

Android applications are released at a rapid pace. Therefore, a relevant and recent dataset is essential for any framework to have any practical use. Data is one of the most important factors in determining the quality of ML models. Unlike many previous studies reported in the literature, we used a balanced, real life and up to date dataset. The applications we used for our dataset were released from 2019 to 2021.

We used VirusShare's [15] most recent Android malware dataset for our malicious applications dataset. We used a total of 20,000 applications from VirusShare. For our benign dataset, we crawled several Android repositories including UpToDown [16], APKMirror [17] and F-Droid [18]. In total, we downloaded 20,000 benign applications. Each application was labelled using VirusTotal [19] reports. In order to prevent false negatives from leaking into our benign datasets, only applications with zero positives were used for the benign dataset.

We ran a static analysis on our dataset to collect the API calls used by each application and built a Boolean dataset. Let $A = \{API_1, API_2, API_3, \ldots API_n\}$ be the complete API set consisting of a total of n number of API calls. Each application's

attribute in the Boolean dataset is set A plus the label. We used 0 to indicate the application does not use the API and 1 indicates that the API is used and label is set to 0 for benign and 1 for malware. For example, if $A = \{API1, API2, API3\}$ and an application used $API1$ and $API3$ and is a malware, the vector of this application will be $M = \{1, 0, 1, 1\}$.

2.2 ML Models

We trained models using Support Vector Machine (SVM), Decision Tree (DT), Random Forest (RF), Naïve Bayes (NB) and AdaBoost on all the feature sets including the complete API calls feature set and the subsets produced by feature selection methods. These are all standard ML models, but for reference we include a brief description of each:

- **Support Vector Machines (SVM)** find the optimal hyperplane that separates the samples from two classes in their n-dimensional feature space. SVMs can also solve non-linear problems by using a kernel trick to project the data into a higher-dimensional space, but here we use an SVM with a linear kernel. SVMs are known for their speed and robustness.
- **Decision Trees (DT)** are tree-structured decision-making processes. Each node in the tree considers a single feature, and based on its value, passes control to one of its two sub-branches. When it reaches the leaf nodes of the tree, a sample is assigned to a particular class. DTs are relatively interpretable ML models, but the use of perpendicular decision planes can limit their accuracy.
- **Random Forest** is another tree-based classifier which uses many DTs to improve the accuracy of single DT models. Random forests train multiple DTs and then output the majority classification. The number of trees we used in our models was 500.
- **Naïve Bayes** is a simple probabilistic classifier model based on Bayes' theorem. It can be used for both classification and regression. Training and testing a Naïve Bayes classifier is comparatively fast in comparison to other ML models, which allows it to scale to large data sets.
- **AdaBoost,** or adaptive boosting, is a classic ensemble learning approach that combines multiple weak classifiers into one relatively strong classifier. For our models we used DT as the base classifier, with maximum number of estimators set to 50.

We evaluate the performance of the models using 10-fold cross-validation (CV) on our dataset. We report the mean and standard deviation of the accuracy, precision, F1-score, true positive rate (TPR) and true negative rate (TNR) to provide a complete picture of how the model performs.

2.3 Feature Selection Methods

Feature ranking is used in machine learning to measure the relevance of the feature to its class label. This helps in selecting the most relevant and informative features in order to improve the model's performance [20]. We used three feature selection algorithms to reduce the number of features from the API calls feature vector: mutual information, variance threshold and PCC.

- **Mutual Information** is the measure of information obtained between two random variables. The value of mutual information is always greater than or equal to zero. Two variables are independent when the value is zero, and, the greater the value, the stronger their relationship is. We calculated the mutual information of all the features in our dataset. Features with the highest mutual information were then used to train the ML models.
- **Variance Threshold** measures the variance of each feature within a dataset, and then eliminates those which have a variance below a specified threshold. It is based on the premise that features which have similar values within different samples tend not to be useful for classification. For instance, in the extreme case where a feature has zero variance, the feature's value is the same for every sample, and therefore provides no information. We calculated the variance of each feature in our dataset. We then used the features with highest variance to train the ML models.
- **Pearson Correlation Co-efficient (PCC)** is used to calculate how linearly dependent a variable is to its target label. The resulting coefficient is greater or less than 0 if the two variables are related, whereas the coefficient is 0 if there is no correlation. We calculated the PCC of all the features in our dataset. The features with the highest PCC were then used to train the ML models.

3 Experimental Results and Analysis

In this section we discuss the ML models and feature selection methods used and report the results from each experiment.

Table 1 Average and standard deviation of evaluation metrics of full API feature set trained on five ML models using 10-fold CV

Classifier	Accuracy	Precision	F1-Score	TPR	TNR
SVM	0.955±0.002	0.957±0.004	0.955±0.002	0.953±0.004	0.957±0.004
DT	0.940±0.000	0.938±0.003	0.941±0.002	0.943±0.002	0.938±0.002
Random Forest	**0.959**±0.001	**0.960**±0.001	**0.959**±0.002	**0.957**±0.001	**0.960**±0.002
Naïve Bayes	0.744±0.002	0.673±0.003	0.789±0.002	0.957±0.002	0.531±0.005
AdaBoost	0.943±0.002	0.943±0.001	0.943±0.002	0.944±0.004	0.942±0.001

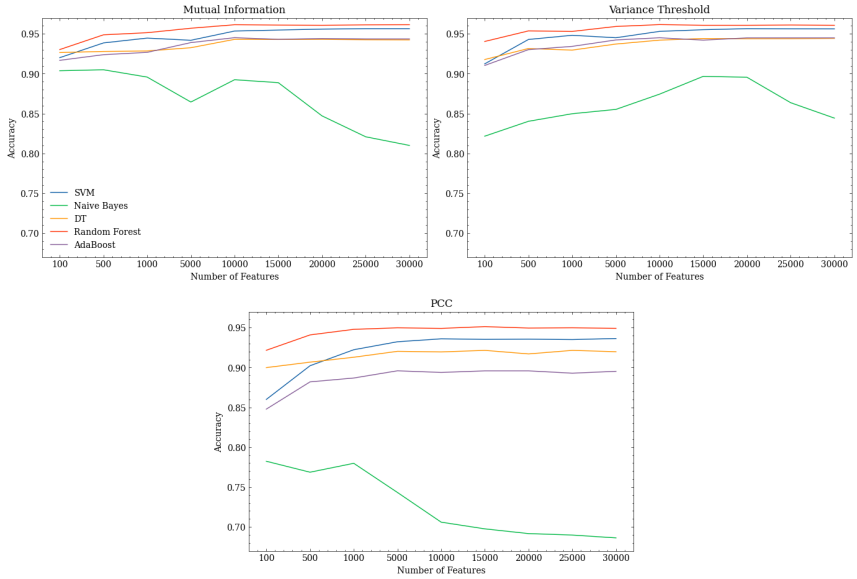

Fig. 1 Mean accuracy of models trained on features selected using mutual information, variance threshold and PCC using 10-fold CV

Table 1 shows the results of applying the five ML models to the complete feature set, which comprises 134,207 API calls. It can be seen that random forest achieved the best results in all the metrics, followed by SVM. AdaBoost performed slightly better than DT. Although Naïve Bayes took the least time to train, the accuracy of the model was very low.

We then applied the feature selection algorithms, in order to reduce the number of API calls used for classification. Specifically, the three feature selection algorithms were used to reduce the feature set to sizes between 100 and 30,000 API calls. The results are shown in Fig. 1 . This shows that the feature set size does have a significant affect on the accuracy of the ML models. However, beyond a certain feature set size threshold, for most of the models the accuracy approaches those achieved using the full feature set.

Random forest remains the most accurate classifier, regardless of the feature set size. Table 2 shows the detailed metrics for the best performing random forest model, for each of the feature selection methods. The overall best models are found when variance threshold and mutual information are used, with PCC resulting in significantly poorer models. Notably, the model produced after variance thresholding is better in all metrics than the model produced from the full feature set, despite using only 5% of the features.

In general, there was a considerable reduction in time taken to train the ML models on the reduced feature sets. This however did not affect the model accuracy considerably. In fact, SVM for the most part maintained the accuracy it achieved with the full API feature set and even reported higher accuracy with mutual information

Table 2 Average and standard deviation of evaluation metrics of the best performing ML models using top features from feature selection algorithms

Feature Selection	Classifier	No. of Features	Accuracy	Precision	F1-Score	TPR	TNR
Mutual Information	Random Forest	10,000	0.962±0.001	0.960±0.001	0.961±0.002	0.959±0.002	0.963±0.002
Variance Threshold	Random Forest	6,443	0.961±0.002	0.964±0.001	0.961±0.002	0.958±0.002	0.964±0.001
PCC	Random Forest	15,000	0.951±0.001	0.952±0.002	0.951±0.002	0.938±0.001	0.965±0.002

and variance threshold. AdaBoost only performed better than DT in some feature set sizes when using mutual information and variance threshold, and performed worse than DT in PCC. Naïve Bayes showed an improvement in accuracy with less number of features only when mutual information and variance threshold were used. The accuracy rate increased from 74.4% using full feature set to 90.5% when using mutual information. However, the accuracy dropped considerably as the number of features was increased.

We also observed that there is a high degree of overlap between the features selected by variance threshold and those selected by mutual information. The top 5,000 features by variance threshold and mutual information shared 3,958 features and the top 10,000 features from both shared 8,598 features. On the other hand, there were only 344 preserved features present in the top 10,000 and 2,278 in the top 20,000 lists produced by mutual information and PCC. The top 20 features ranked by mutual information and variance threshold are provided in Table 3.

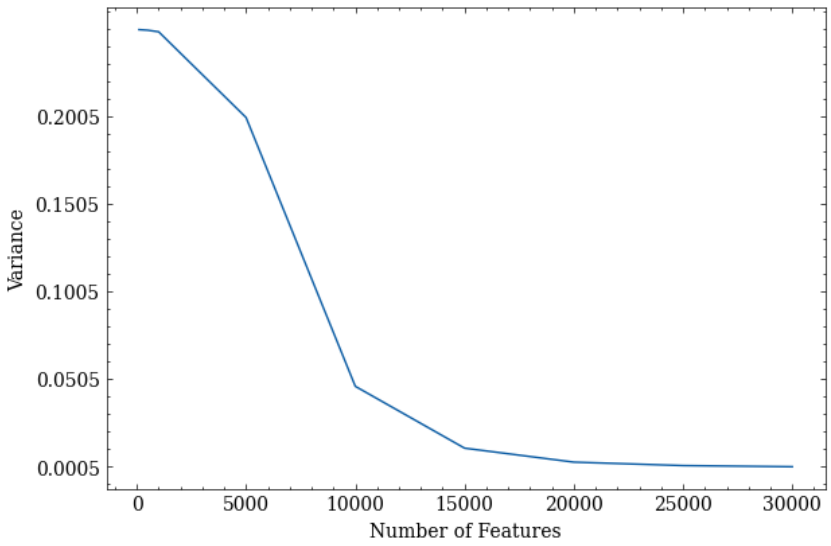

Fig. 2 Number of features selected by variance thresholds

Fig. 2 shows the variance values for the feature set sizes of 100 to 30,000. A substantial drop in variance after 5,000 features can be seen from the plot. The fall in

variance signifies that many of the features do not greatly impact the accuracy of the models. Therefore, the accuracy rates for mutual information and variance threshold increased considerably only up to 10,000 features, coinciding with the plot in Fig. 2, where there is a marked decrease in variance after 10,000 features. We further investigated the best threshold value between 0.05 and 0.25 to identify the variance threshold that produces the best accuracy metrics. The threshold of 0.15 achieved the highest accuracy with 6,443 features.

Table 3 Top 20 features ranked by mutual information and variance threshold

Rank	Mutual Information	Variance Threshold
1	Landroid/content/pm/PackageInstaller$SessionInfo;->getAppPackageName	Landroid/content/res/Resources$Theme;->applyStyle
2	Landroid/content/pm/PackageInstaller;->getAllSessions	Landroid/app/Notification$Builder;->setCustomHeadsUpContentView
3	Landroid/os/DeadObjectException;-><init>	Landroid/view/accessibility/AccessibilityNodeInfo;->setViewIdResourceName
4	Landroid/os/UserManager;->getApplicationRestrictions	Landroid/content/Context;->getObbDirs
5	Landroid/content/pm/PackageManager;->getPackageInstaller	Landroid/media/MediaPlayer;->setOnErrorListener
6	Landroid/os/Binder;->restoreCallingIdentity	Landroid/graphics/drawable/LayerDrawable;->setId
7	Landroid/os/Binder;->clearCallingIdentity	Landroid/content/pm/PackageManager;->queryIntentActivityOptions
8	Landroid/os/IInterface;->asBinder	Landroid/os/Parcel;->writeValue
9	Landroid/os/Parcel;->dataPosition	Landroid/widget/ProgressBar;->setIndeterminate
10	Landroid/app/ActivityManager;->getMyMemoryState	Landroid/widget/ListView;->setOnKeyListener
11	Landroid/content/ServiceConnection;->onServiceConnected	Landroid/text/Editable;->length
12	Landroid/content/ServiceConnection;->onServiceDisconnected	Landroid/view/accessibility/AccessibilityNodeInfo;->setContentInvalid
13	Landroid/app/AppOpsManager;->checkPackage	Ljava/io/FileNotFoundException;->printStackTrace
14	Landroid/content/pm/PackageManager;->isInstantApp	Landroid/content/Context;->getObbDir
15	Landroid/os/PowerManager;->isInteractive	Landroid/app/Notification$Builder;->setCustomContentView
16	Landroid/os/Parcel;->dataSize	Landroid/view/MenuItem;->getIcon
17	Ljava/util/logging/Logger;->logp	Landroid/database/Cursor;->getExtras
18	Ljava/lang/Character;->isSurrogatePair	Landroid/hardware/display/DisplayManager;->getDisplays
19	Ljava/util/TreeMap;->descendingMap	Landroid/app/Notification$Builder;->setCustomBigContentView
20	Landroid/view/ViewGroup;->startViewTransition	Landroid/os/ResultReceiver;-><init>

4 Conclusion

As the popularity of the Android OS has increased in recent years, it has become a major target for malware developers. This can result in personal data becoming vulnerable and calls for developing robust anti-malware techniques. Research shows machine learning techniques work well in identifying new and old malware.

API calls is one of the most used features in Android malware detection using machine learning. However, due to the number of APIs provided by the Android SDK, the number of API calls used by applications can become overwhelming from a machine learning perspective. In our dataset of 40,000 applications, 134,207 different API calls were used. In this work, we analysed how different ML models, namely SVM, decision tree, random forest, Naïve Bayes and AdaBoost, perform with the API feature set and how the API feature set can be reduced for more practical use with three feature selection algorithms, mutual information, variance threshold and Pearson Correlation Co-efficient. The results show that random forest classifiers perform the best when used with an API calls feature set, and we can reduce the number of features by 95.2% to achieve better detection accuracy than the complete API call feature set.

References

1. A. Turner, "How many smartphones are in the world?" https://www.bankmycell.com/blog/how-many-phones-are-in-the-world, 2021, [Online accessed July 31, 2021].
2. StatCounter, "Mobile operating system market share worldwide," 2021, [Online; Accessed: April 16, 2021]. [Online]. Available: http://gs.statcounter.com/os-market-share/mobile/worldwide
3. "Google play store," https://play.google.com/store.
4. C. Osborne. (2021) Joker billing fraud malware found in google play store. https://www.zdnet.com/article/joker-billing-fraud-malware-found-in-google-play-store/. Online accessed July 31, 2021.
5. B. Yu, Y. Fang, Q. Yang, Y. Tang, and L. Liu, "A survey of malware behavior description and analysis," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 5, pp. 583–603, 2018.
6. N. Peiravian and X. Zhu, "Machine Learning for Android Malware Detection Using Permission and API Calls," in *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*. IEEE, nov 2013, pp. 300–305. [Online]. Available: http://ieeexplore.ieee.org/document/6735264/
7. D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," in *Network and Distributed System Security Symposium (NDSS)*, no. August, 2014.
8. 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, no. c, pp. 21 235–21 245, 2019.
9. J. Jung, H. Kim, D. Shin, M. Lee, H. Lee, S. J. Cho, and K. Suh, "Android Malware Detection Based on Useful API Calls and Machine Learning," *Proceedings - 2018 1st IEEE International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2018*, pp. 175–178, 2018.
10. 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," *Journal of Computer Virology and Hacking Techniques*, vol. 11, no. 1, pp. 9–17, 2015.
11. X. Xiao, S. Zhang, F. Mercaldo, G. Hu, and A. K. Sangaiah, "Android malware detection based on system call sequences and LSTM," *Multimedia Tools and Applications*, vol. 78, no. 4, pp. 3979–3999, 2019.
12. A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, 2018.
13. "Package index," https://developer.android.com/reference/packages, 2021, [Online accessed July 31, 2021].
14. R. Connor Tumbleson, "Apktool," https://ibotpeaches.github.io/Apktool/, 2019.
15. "Virusshare," https://virusshare.com.
16. "App downloads for android," https://en.uptodown.com/android, [Online accessed July 31, 2021].
17. "Apkmirror," https://www.apkmirror.com/, [Online accessed July 31, 2021].
18. "F-droid - free and open source android app repository," https://www.f-droid.org/, [Online accessed July 31, 2021].
19. "Virustotal," https://www.virustotal.com.
20. I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, no. null, p. 1157–1182, Mar. 2003.