Prediction of Android Malicious Software Using Boosting Algorithms

Deepon Deb Nath[1], Nafiz Imtiaz Khan[2(✉)], Jesmin Akhter[3], and Abu Sayed Md. Mostafizur Rahaman[4]

[1] Department of Information and Communication Technology, Bangladesh University of Professionals (BUP), Dhaka, Bangladesh
[2] Department of Computer Science and Engineering, Military Institute of Science and Technology (MIST), Dhaka, Bangladesh
nafiz@cse.mist.ac.bd
[3] Institute of Information Technology, Jahangirnagar University, Dhaka, Bangladesh
[4] Department of Computer Science and Engineering, Jahangirnagar University, Dhaka, Bangladesh

Abstract. Android malware, a group of malicious software variants, including viruses, ransomware and spyware, designed to cause substantial damage to data and systems or to access a network without authorization. With an inexorable shift in technology, Android has supplanted other Mobile platforms by being flexible and user-friendly to the users. As the number of Android apps continues to grow every day, the number of malwares aimed at attacking those users is also on the rise. Thus, it becomes emergent to identify and remove malicious Android applications before installation to prevent user's loss. Several studies have already been carried out to anticipate Android malware using machine learning algorithms, while as per the literature survey conducted by this study, a significant research has not been found to be focusing especially on the genre of boosting algorithms. Therefore, the objective of this paper is to classify malicious and benign Android applications by using Boosting algorithm. To attain the research objective, four widely defined boosting models viz. AdaBoost, CatBoost, XGBoost, and GradientBoost were developed whereas, it was found that CatBoost and GradientBoost had the highest F1 score (93.9%), followed by Adaboost (F1 score 93.5%), and XGBoost (F1 score 93.5%).

Keywords: Android · Malware · Classification · Boosting algorithms · Machine learning · Static analysis

1 Introduction

Android has proven to be the world's most widely used intelligent terminal operating system because of its numerous benefits: open-source, extensibility, and convenience. According to [1] in May 2021, Android has a 72.72% market share worldwide among mobile operating systems. Therefore, much mobile malware is likely to continue to be developed and distributed to execute various cybercrimes on mobile phones. As of

M. H. Miraz et al. (Eds.): iCETiC 2021, LNICST 395, pp. 21–36, 2021.
https://doi.org/10.1007/978-3-030-90016-8_2

March 2020 in [2], 482,579 new Android malware samples amounted per month. These malicious apps are mostly distributed through third-party markets, but even Google Android Market cannot assure the applications that it registers into it are virus-free. Malicious apps pose a much more dangerous security risk to users. Spyware, Phishing, Bots, RootExploits, Banking-Trojans, Premium Dialers, SMS Fraud, and other threats are among them. While Android apps are handy for users, private information and vital data (such as information about medical records, bank account, credit card information, and passwords, etc.) are continually under threat as well.

To assure the safety of the Android system, various malware detection techniques have been recommended. Malware detection methods are mainly classified into three types: static detection method, dynamic detection method, and hybrid detection method [3, 4]. The first detection methods take out syntactic features that can be observed when an application is run in a controlled environment, whereas the second methods take out semantic features that can also be observed when an application is run in a controlled environment. This can reveal risks that static analysis cannot, but the time, cost, and computational resources of dynamic detection are comparatively high. Finally, The third method "hybrid detection" refers to a method that combines the first and second methods to accomplish a balance of detection efficiency and efficacy.

As of today, the signature-based detection technique is the most widely utilized. In this method, Experts manually define the signatures of malware and afterwords the malware is identified by looking for signatures in apps. Malware authors can make use of obfuscation technology to change malware's signature, granting malware to simply evade recognition by detection engines which are one of the disadvantages of the signature-based technique. To address this problem, an intelligent malware recognition technique is proposed. Machine learning algorithms are used by the intelligent malware detection technique to identify malware whether the technique is dynamic, static, or hybrid. Hidden patterns in malware can be learned by machine learning algorithms. These patterns have a high degree of generalization and can differentiate between benign and malicious applications. Unlike traditional approaches of malware detection, such as signature-based technique, machine learning (ML) based detection can identify previously undetected types of malware [5] and can deliver improved detection efficiency and efficacy [6].

The primary goals of this research are to examine various boosting algorithms for predicting android malware and to identify the relative feature importance of each features for developing ML models. The structure of the paper is as follows. The literature review on Android malware detection is described in Sect. 2. Section 3 discusses the overall research methodology, while the experimental results, analysis and evaluations are descried in Sect. 4. Finally, the discussion, conclusion and the future plan are narrated in Sect. 5.

2 Literature Review

Machine learning algorithms can be used to learn the most essential patterns from malware samples which can distinguish between dangerous and benign applications. Various ML algorithms such as Support Vector Machines (SVM), Naive Bayes method, Decision Trees, and Random Forest (RF) have been employed in this field to detect unknown

harmful executables. These studies demonstrate the effectiveness of machine learning methods in detecting unknown malware.

To illustrate the features of malware which is commonly stated as a feature vector is one of the key tasks of an ML-based malware recognition system. If the feature vector's dimension is very huge, feature selection techniques can be used to lower it's dimension. For extracting behavioral features, malware samples are usually run in a virtual environment or sandbox. Mohaisen et al. [7] studied the behavior of malware illustrations in a simulated environment. Following that, they selected program behaviors such as network activities, file operations, memory operations, and registry key changes to depict malware. Besides, They used K-Nearest-Neighbor (KNN), Linear Regression (LR), the Perceptron, and SVM to categorize malware. Moreover, they extracted 65 features from malware to represent it. Pirscoveanu et al. [8] extracted 151 API calls which were used to characterize malware by running malware samples in the Cuckoo sandbox and classified them using the random forest classifier.

Malicious applications were detected with unauthorized permission attacks by Ankita [9] using 103 malware and 97 benign applications datasets, respectively on Nexus 5 with API level 19. Simple logic, J.48, RF 100, Naive Bayes, RF 10, IBK algorithms, and Sequential minimal optimization were used in the experimental approach. Here, The permission request was extracted by the XML parser, which generated binary malware features that were saved in the Attribute Relation File Format (ARFF). When the random forest algorithm was used, the detection rate was 96.6%. The researchers in [10] proposed a model for monitoring application events, integrating access, modifying permission levels into a critical function (RF) and sensitive APIs to protect users from the high levels of destruction caused by Android malicious applications. A database of 2130 samples is being used to specifically test the effectiveness of the proposed technique. According to experimental results, the proposed method accomplishes a high precision of 88.26%, 88.40% sensitivity, and 88.16% accuracy.

The dynamic detection technique [11] was tested on 4034 malware datasets and 10024 benign datasets. The random forest classifier detected malware on those applications with 96% accuracy while using the ServiceMonitor method. Here features attributes are extracted by the classifier module and this module is trained by using k-fold validation and the Markov chain. Malware retrieved information, like phone IMEI was found with an accuracy of 67%. In interchange for a better service rating, 17% of the detected malicious applications have attached their payload to the device.

Some malware remains dormant on the device after it has been downloaded and installed until an action is taken. Others, on the other hand, execute their payload during download, installation, and runtime [12]. Despite the fact that default permissions are continually encountered during install and download sessions, the access authorization routinely granted by Android users creates a huge space in the device attack vector. Malicious code attaches itself to benign applications during these exercises. At these stages, critical monitoring is essential for improved mobile platform security.

Based on deep learning, a static detection method [13] for packed or encrypted malware has been proposed. From the Android APK file, Bytecode files can be extracted and then converted into a two-dimensional bytecode matrix, and finally used to train and categorize malware using the deep learning algorithm, convolution neural network

(CNN). CNN learned features of bytecode files spontaneously that can be used to categorize malicious applications. The suggested detection method avoids the steps of investigating malware features and crafting malware feature illustrations.

To improve the accuracy and efficacy of comprehensive Android malware detection, the authors in [14] suggested a hybrid model based on convolutional neural networks (CNN) and deep autoencoders (DAE). Here neural networks demonstrate a strong ability to detect malware with 99.80% accuracy. While compared to the CNN-S model, the DAC-CNN model also reduces training time by 83%. To identify zero-day botnet attacks, the deep learning-based model for detecting botnets in real-time has been proposed by Ahmed et al. [15]. In 2019, Ding and Zhu [16] characterize malware as opcode sequences and used a deep belief network (DBN) to extract malware features in an analogous manner. Their experiments demonstrate that the DBN model outperforms baseline models such as the k-nearest neighbor algorithm, SVM, and decision trees as classifiers.

The hybrid detection approach incorporates the advantages of both dynamic and static detection methods to provide strong detection outcomes While analyzing malicious applications. Because the strengths of both methods are combined, it appears that this method has a better detection percentage than static and dynamic methods. A Droid-Detector model [17] for android malware recognition was designed and trained utilizing artificial intelligence's deep learning capability [18]. This model produced detection outcomes with a 96.6% accuracy and a 0.0021% distinction among the algorithms applied. Hybrid technology also allows for precise comparison of static and dynamic detection rates. Using a hidden Markov model [19] and the semantic approach of this method, a hidden example sequence was extracted from job code and API calls. The ROC curve's threshold was determined by reproducibility, precision, and specificity. The Android Buster sandbox was used as an analysis tool to define and determine the application's maliciousness and positivity. However, using the API call sequence to detect android malware does not solve the malware obfuscation problem.

To avoid infecting other network devices, the emulation-based detection tech nique necessarily requires the creation of sandboxes and the configuration of virtual machines systematically and securely. This technique is particularly effective when the Dalvik file (.dex) [20] is properly monitored. When malware is executed in the mobile real OS, detection turn out to be very much difficult. Malicious applications can be recognized in the sandbox environment by getting the dex file and transforming it into a human-readable format. This technique effectively captures zero-day malware [21] and malware that escalates privileges [22]. Sometimes, malware understands the environment's virtual nature and attempts to avoid recognition.

In addition to employing dynamic monitoring tools to achieve malware feature interpretation, decompiling tools such as IDAPro and Captone can be used to obtain the behavior features of malware. Cesare et al. [23] used decompilers to extract a program's control flow graph and identified malware variants by associating control flow graph similarity. Chan and Song [24] suggested a feature set for Android malicious application recognition that incorporated API calls and permissions. Wang et al. [25] categorized malware using application and platform specific static features using linear SVM, logistic regression, Random Forest, and Decision Tree. In paper [26], the authors developed

sensitive subgraphs to illustrate static features demonstrating invocation patterns. Classifiers such as Decision Tree, Random Forest, PART, and K-Nearest Neighbor (KNN) were trained using the features, with Random Forest outperforming the others. Optimizing the hyper-parameters of a single class support vector machine used to identify an IoT botnet using the Grey Wolf Optimization swarm intelligence algorithm is described in [27].

In the field of android malware recognition, the N-gram technique has also been analyzed and evaluated in combination with stacked generalization. To assess the usefulness of unigram, bigram, and trigram along with stacking, a detailed analysis was performed in [28]. It has been discovered that when combined with stacking, unigram provides more than 97% accuracy, the highest detection rate while compared to others.

3 Research Methodology

The overview of research methodology is shown in Fig. 1, which includes data acquisition, feature selection, data synthesis, data oversampling, models development, and prediction. The stages are briefly discussed in the following subsections.

3.1 Data Acquisition

The dataset used in the experiment is from the University of Gottingen's Drebin project [29]. 215 features were obtained from 15,036 applications in this dataset (9,476 benign and 5,560 malware). Drebin samples are broadly used in the scien tific community. This dataset was also employed to create and assess a multilevel classifier fusion method for identifying Android malware [30].

3.2 Feature Selection

To lower the dimension of the chosen dataset, the Recursive Feature Elimination (RFE) method was applied. RFE works by deleting features in a recursive manner and then creating a model on the remaining attributes. It determines the combinations of attributes that contribute the most to predicting the target attribute by looking at the model accuracy. In this study, RFE was applied with a Logistic Regression algorithm to select the top 20 features, which are shown in Table 1, for predicting the target variable (type of application).

3.3 Data Synthesis

In this phase, the reduced feature set was processed to feed to the ML mod els. First, the dataset's missing values were addressed; the mean values were substituted for missing numerical cells, whereas the most frequent values were substituted for missing category cells. Second, the target feature was encoded as it was labeled as 'M', short of malware, and 'B', short of Benign. Third, a random train-test split of 75–25 was applied to the dataset, where 75% data was considered as train set while 25% data was considered as the test set.

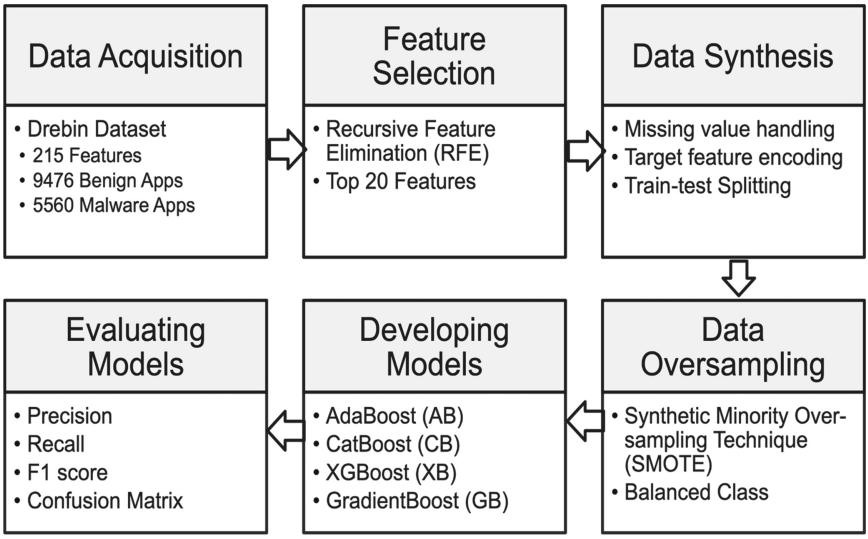

Fig. 1. Overview of research methodology

3.4 Data Oversampling

Drebin dataset had a class imbalance issue [31] as there were 9476 samples labeled as benign while 5560 samples labeled as malignant. It is required to have a similar number of instances for every class so that the ML models do not get biased toward a particular class. To handle the class imbalance problem, the Synthetic Minority Over-sampling Technique (SMOTE) [32] was used as the over-sampling procedure, which removes the imbalances in the classes. This method imitates current minority data instances and makes modest modifications to them, resulting in new minority data instances. Both the malignant and benign classes have exactly 9476 samples after the oversampling technique.

3.5 Developing Models

In this phase, four different ML models were developed by utilizing four algorithms from the domain of boosting algorithms. The algorithms subsume AdaBoost, CatBoost, XGBoost, and GradientBoost. The models were developed by using Python programming language and Sci-kit learn implementation package. The following is a brief overview of the considered algorithms.

1. **AdaBoost** AdaBoost which stands for Adaptive Boosting is a meta-learning ML algorithm, which is used to ameliorate the performance of the model by fitting sequentially to the weak learning models such as Decision Trees [33]. The output of the weak learners is merged into a weighted sum that reflects the final output of the boosted classifier. Because of the strong tugging of weak learners, when examples are misclassified, this classifier is adaptive. It is less prone to overfitting than other learning algorithms, yet being sensitive to noisy input [33].

Table 1. Reduced set of features (n = 20)

SL.	Feature name
1	Send_Sms
2	Ljava.lang.Class.getCanonicalName
3	getCallingUid
4	Use_Credentials
5	Manage_Accounts
6	android.intent.action. send
7	android.telephony.gsm.SmsManager
8	Read_History_Bookmarks
9	android.intent.action.Package_Replaced
10	nfc
11	Bind_RemoteViews
12	TelephonyManager.getDeviceId
13	Modify_Audio_Settings
14	android.intent.action.Timezone_Changed
15	chmod
16	Runtime.load
17	Read_Call_Log
18	SendMultipartTextMessage
19	Write_Calender
20	Write_Gservices

2. **CatBoost** CatBoost is a popular machine learning algorithm that handles categorical features efficiently and takes advantage of dealing with them during training rather than pre-processing time. [34]. It reduces the extensive need for hyper-parameter tuning and has a lower chance of overfitting by applying ordered boosting for getting the best result [35].
3. **XGBoost** XGBoost (eXtreme Gradient Boosting) is a boosting technique that is well known for offering parallel tree boosting that is well utilized for tackling data science issues precisely and efficiently in terms of both speed and performance [36]. This method is also frequently used to predict the polarity of online reviews [37] based on consumer purchase choices, where the essential characteristics are retrieved from the data using ranking scores.
4. **GradientBoost** Gradient boosting is another ML technique for solving classification and regression problems [38]. The basis learners are generally decision trees, and this prediction method is essentially an ensemble form of weak prediction models or base learners. A stage-wise fashioned model, like other boosting methods, is optimized using an arbitrary differentiable loss function.

3.6 Evaluating Models

In this phase, the predictions (malware and benign) are obtained from the ML models and their performance was evaluated. This phase is discussed in detail in Sect. 4 (Result Analysis).

4 Result Analysis

The models were evaluated on both training and test data, whereas the algorithms were trained on training data. Precision, recall, and f1 score are used to assess the models' performance. Because of the oversampling approach, the wellknown performance metric'accuracy' was not employed because it is deceptive in this situation [39]. The performance of the models for both the train as well as test data is shown in Table 2. Also, the performance for the train and the test data is shown in graphical form in Figs. 2 and 3 respectively. For each of the models, the feature importance score for each of the features (n = 20) were obtained. The feature importance score delineates that how useful a particular feature is to develop a particular model [40].

Table 2. Performance measures for the developed models

Model	Train			Test		
	Precision	Recall	F1 score	Precision	Recall	F1 score
AB	0.934	0.933	0.933	0.936	0.935	0.935
CB	0.94	0.94	0.94	0.939	0.939	0.939
XB	0.935	0.935	0.935	0.936	0.935	0.935
GB	0.94	0.94	0.94	0.939	0.939	0.939

Considering the AB model, the train precision, recall and f1-score were 93.4%. 93.3% and 93.3% respectively, while the precision, recall and f1-score for the test dataset were 93.6%, 93.5% and 93.5% respectively (see Table 2). The feature importance and the confusion matrices for the AB model are shown in Figs. 5 and 4 respectively. It can be observed from Fig. 5 that, 'getCallingUid', 'android.telephony.gsm.SmsManager' and 'Send sms' were top three features for developing the AB model. nonetheless, for the test dataset, 2197 samples were correctly classified as Benign, while 2252 samples were correctly classified as malignant (see Fig. 4 (b)).

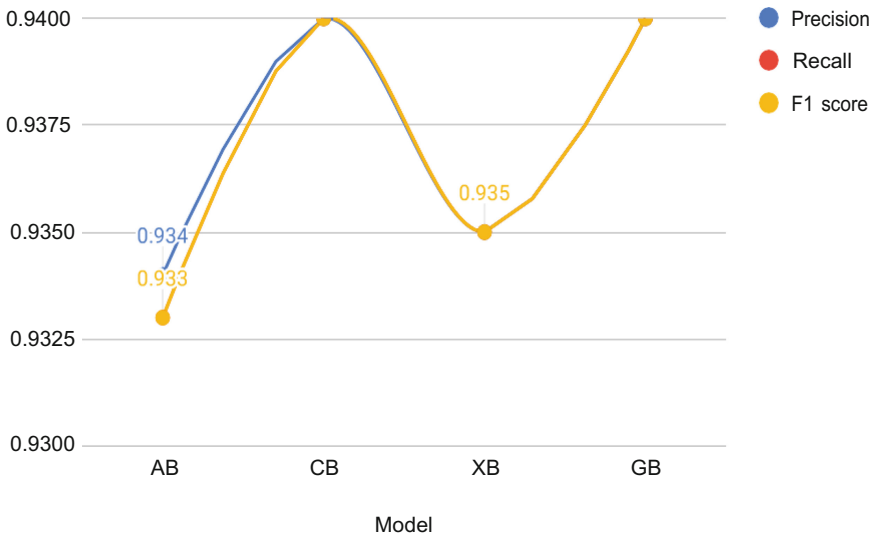

Fig. 2. Performance by developed models on training data

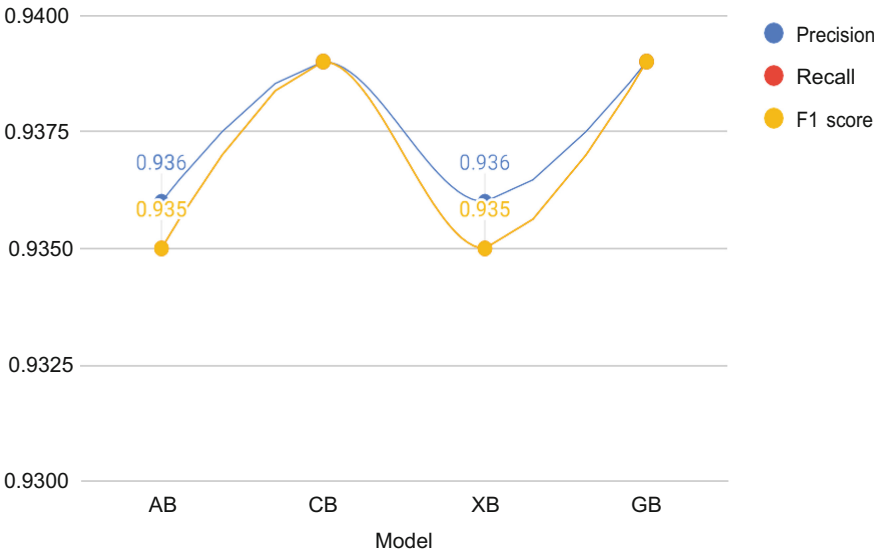

Fig. 3. Performance by developed models on testing data

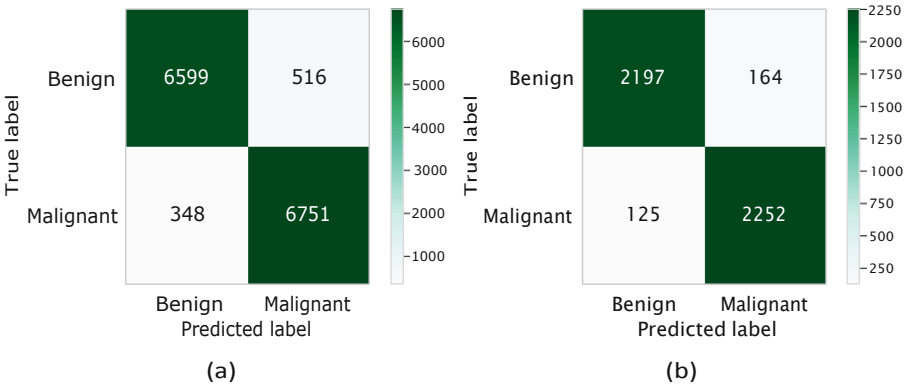

Fig. 4. Confusion matrices of AB classifier for: (a) train data, (b) test data

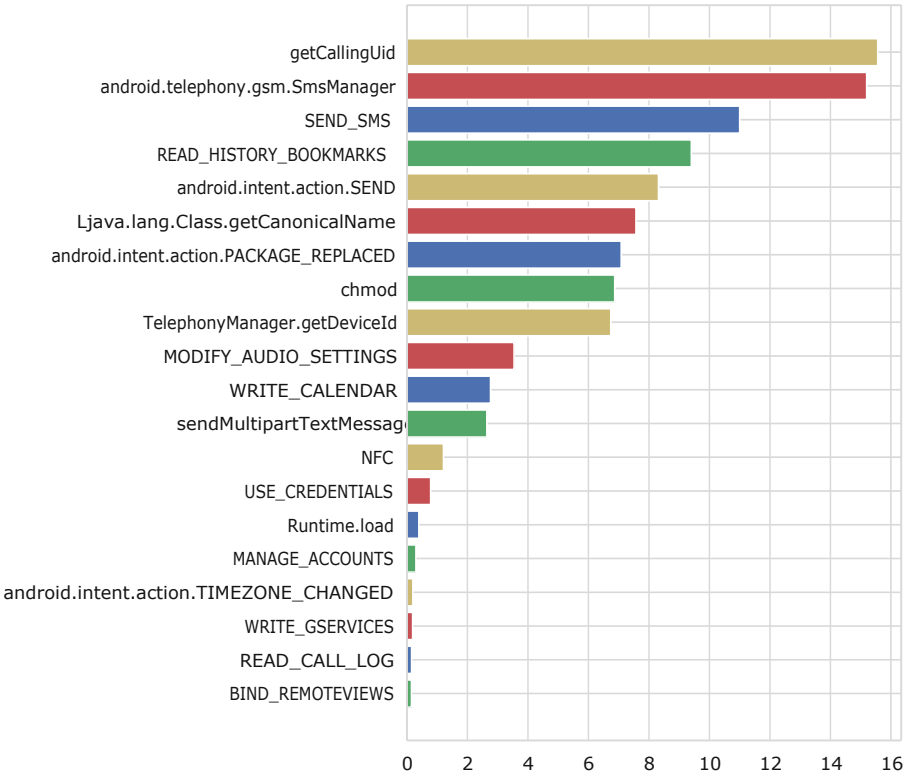

Fig. 5. Feature importance for developing the AB model

For the CB model, precision, recall and f1 score for the train dataset was 94%, while precision, recall and f1 score for the test dataset was 93.9% (see Table 2). The confusion matrices for the CB model are shown in Fig. 6, where for the test dataset, 2197 samples

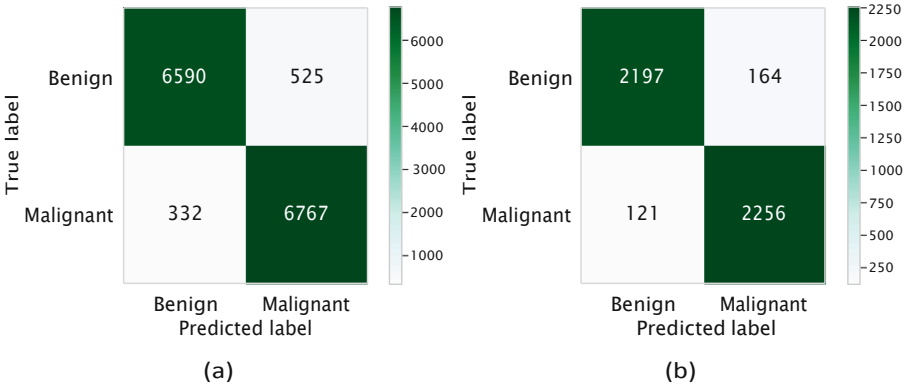

Fig. 6. Confusion matrices of CB classifier for: (a) train data, (b) test data

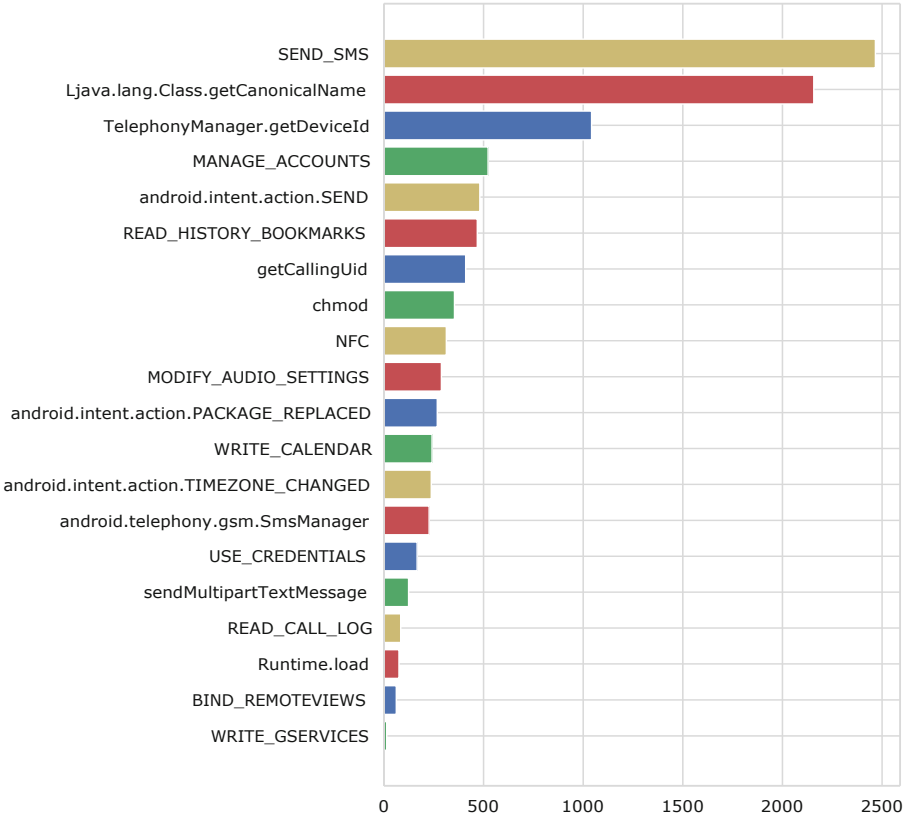

Fig. 7. Feature importance for developing the CB model

were correctly classified as Benign, while 2256 samples were correctly classsfied as Malignant. However, the feature importance for the CB model is shown in Fig. 7. It

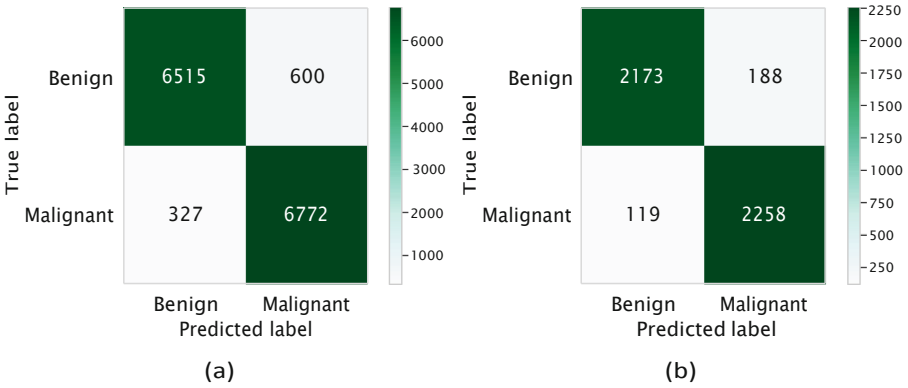

Fig. 8. Confusion matrices of XB classifier for: (a) train data, (b) test data

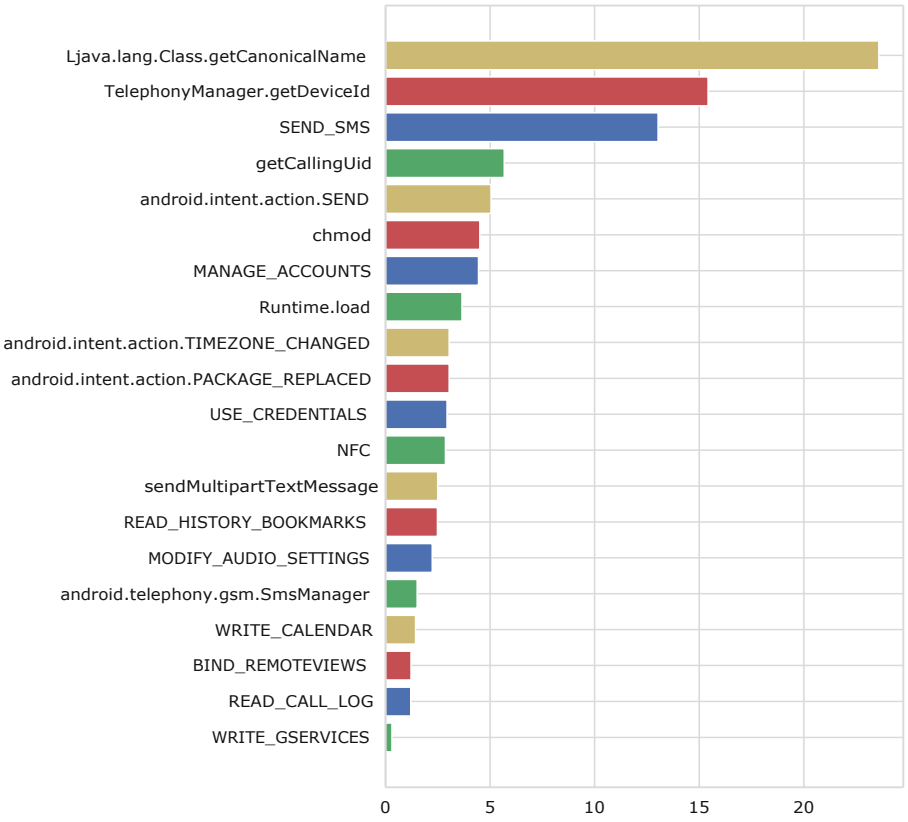

Fig. 9. Feature importance for developing the XB model

can be seen from Fig. 7 that, 'SEND SMS', 'Ljava.lang.class.getCannonicalName' and 'TelephonyManager.getDeviceid' were top three features for developing the CB model.

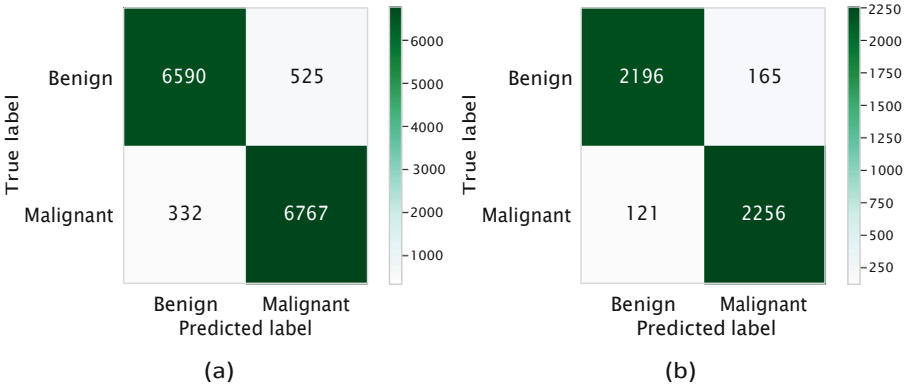

Fig. 10. Confusion matrices of GB classifier for: (a) train data, (b) test data

For the XB model, the precision, recall, and f1 score for the train dataset was 93.5%, while precision, recall and f1 score for the test dataset were 93.6%, 93.5%,

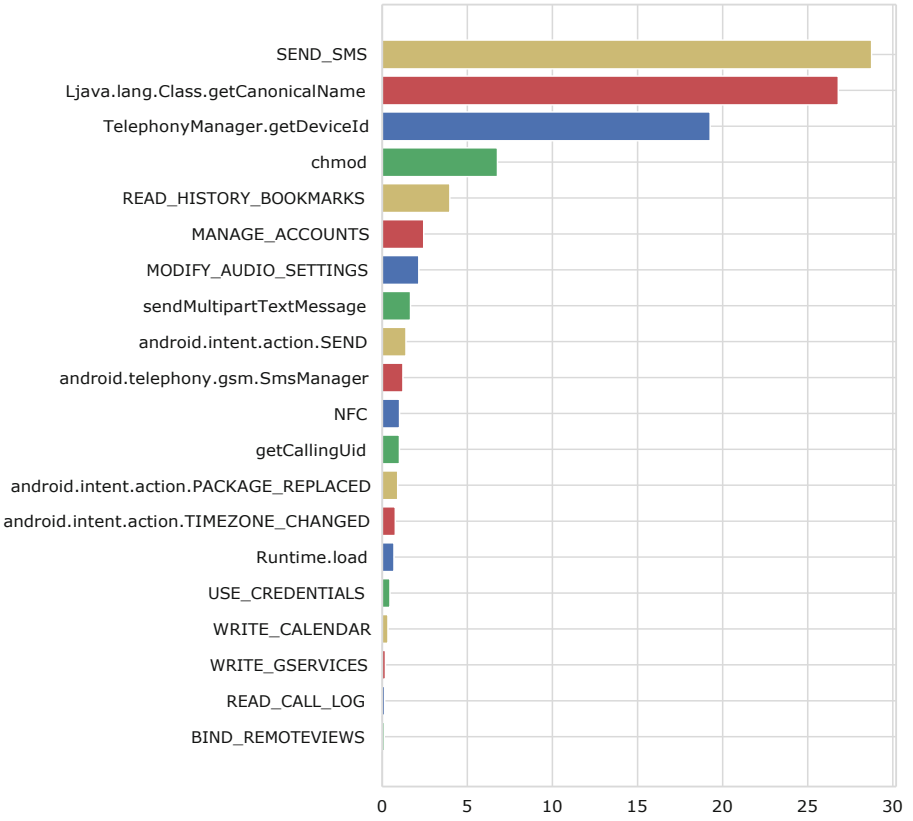

Fig. 11. Feature importance for developing the GB model

93.5% respectively. The feature importance for developing the XB model is shown in Fig. 9. It can be observed that, 'Ljava.lang.Class.getCannonicalName', 'TelephonyManager.getDeviceID', and 'Send sms' were top three features for developing the XB model. Nonetheless, confusion matrices for the XB model are shown in Fig. 8, while it can be observed from the figure that 2173 samples were correctly classified as Benign whereas, 2258 samples were correctly classified as Malignant.

For the GB model, performance metrics (precision, recall, and f1 score) for the train dataset were 94%, while the performance metrics for the test dataset were 93.9%. Confusion matrices for the GB model are shown in figure 10, while 2196 samples were correctly classified as Benign whereas, 2256 samples were correctly classified as Malignant. Figure 11 depicts the feature importance for the GB model, where it can be observed that here also like the XB model, 'Ljava.lang.Class. getCannonicalName', 'Sendsms' and 'TelephonyManager.getDeviceID' were top three features for developing the GB model.

5 Discussion and Conclusions

We propose here a static-based analysis technique for Android malware recognition that employs boosting algorithms and yields three outcomes: First, to predict android malware, top 20 set of features were obtained by utilizing RFE method, Second, four different Boosting ML methods, including AdaBoost (AB), GradientBoost, XGBoost (XB), and CatBoost (CB), were developed on the reduced set of features (n = 20). Third, the relative importance of each feature for developing each of the ML models was determined. The models were trained and tested in order to determine their accuracy, precision, recall, and F1 score. The outcome of this study showed that: among the genre of boosting algorithms, CatBoost and GradientBoost had the highest F1 score (93.9%), followed by AdaBoost (93.5%), and XGBoost (93.5%).

Again, very few studies related to android malware were conducted using the boosting algorithms in the past. While this research showed approximately 93% accuracy for all the developed boosting models while considering only 20 attributes. Considering the outcomes of this study compared to the other published works, this paper has a novel contribution to the research community for predicting malicious android applications. As for limitation, in this work no decompiling tools were employed to analyze malware features, rather the extracted features/attributes from the Drebin dataset were used for classification. Hence, the proposed approach will not be applicable for encrypted or packed android applications. Furthermore, only the algorithms from the category of boosting algorithms were explored. Thus, future studies may work on predicting android malware from encrypted and packed android applications as well as by utilizing algorithms from different domains, such as: traditional, ensemble and deep learning.

References

1. Mobile operating system market share worldwide. https://gs.statcounter.com/os-market-share/mobile/worldwide. Accessed 4 May 2021

2. Development of new android malware worldwide from June 2016 to March 2020. https://www.statista.com/statistics/680705/global-android-malware-volume/. Accessed 4 May 2021
3. Qing, S.H.: Research progress on android security. J. Softw. **27**(1), 45–71 (2016)
4. Lopes, J., Serr˜ao, C., Nunes, L., Almeida, A., Oliveira, J.: Overview of machine learning methods for android malware identification. In: 2019 7th International Symposium on Digital Forensics and Security (ISDFS), pp. 1–6. IEEE (2019)
5. Ahvanooey, M.T., Li, Q., Rabbani, M., Rajput, A.R.: A survey on smartphones security: software vulnerabilities, malware, and attacks. *arXiv preprint* arXiv:2001.09406 (2020)
6. Souri, A., Hosseini, R.: A state-of-the-art survey of malware detection approaches using data mining techniques. HCIS **8**(1), 1–22 (2018). https://doi.org/10.1186/s13673-018-0125-x
7. Mohaisen, A., Alrawi, O., Mohaisen, M.: Amal: high-fidelity, behavior- based automated malware analysis and classification. Comput. Secur. **52**, 251–266 (2015)
8. Pirscoveanu, R.S., Hansen, S.S., Larsen, T.M., Stevanovic, M., Pedersen, J.M., Czech, A.: Analysis of malware behavior: type classification using machine learning. In: 2015 International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), pp. 1–7. IEEE (2015)
9. Kapratwar, A., Di Troia, F., Stamp, M.: Static and dynamic analysis of android malware. In: ICISSP, pp. 653–662 (2017)
10. Zhu, H.-J., You, Z.-H., Zhu, Z.-X., Shi, W.-L., Chen, X., Cheng, L.: Droiddet: effective and robust detection of android malware using static analysis along with rotation forest model. Neurocomputing **272**, 638–646 (2018)
11. Salehi, M., Amini, M.: Android malware detection using Markov chain model of application behaviors in requesting system services. *arXiv preprint* arXiv:1711.05731 (2017)
12. Mahindru, A., Singh, P.: Dynamic permissions based android malware detection using machine learning techniques. In: Proceedings of the 10th In novations in Software Engineering Conference, pp. 202–210 (2017)
13. Ding, Y., Zhang, X., Hu, J., Xu, W.: Android malware detection method based on bytecode image. J. Ambient Intell. Humaniz. Comput. 1–10 (2020). https://doi.org/10.1007/s12652-020-02196-4
14. Wang, W., Zhao, M., Wang, J.: Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. J. Ambient Intell. Humaniz. Comput. **10**(8), 3035–3043 (2018). https://doi.org/10.1007/s12652-018-0803-6
15. Ahmed, A.A., Jabbar, W.A., Sadiq, A.S., Patel, H.: Deep learning-based classification model for botnet attack detection. J. Ambient Intell. Humaniz. Comput. 1–10 (2020). https://doi.org/10.1007/s12652-020-01848-9
16. Yuxin, D., Siyi, Z.: Malware detection based on deep learning algorithm. Neural Comput. Appl. **31**(2), 461–472 (2017). https://doi.org/10.1007/s00521-017-3077-6
17. Yuan, Z., Yongqiang, L., Xue, Y.: Droiddetector: android malware characterization and detection using deep learning. Tsinghua Sci. Technol. **21**(1), 114–123 (2016)
18. Demetrio, L., Biggio, B., Lagorio, G., Roli, F., Armando, A.: Explaining vulnerabilities of deep learning to adversarial malware binaries. *arXiv preprint* arXiv:1901.03583 (2019)
19. Damodaran, A., Di Troia, F., Visaggio, C.A., Austin, T.H., Stamp, M.: A comparison of static, dynamic, and hybrid analysis for malware detection. J. Comput. Virol. Hacking Tech. **13**(1), 1–12 (2017)
20. Costa, G., Aria, H.: Android malware detection using network behavior analysis and machine learning classifiers (2017)
21. Ashawa, M.A., Morris, S.: Analysis of android malware detection techniques: a systematic review (2019)
22. Lee, H.-T., Kim, D., Park, M., Cho, S.: Protecting data on android platform against privilege escalation attack. Int. J. Comput. Math. **93**(2), 401–414 (2016)

23. Cesare, S., Xiang, Y., Zhou, W.: Control flow-based malware variant-detection. IEEE Trans. Dependable Secure Comput. **11**(4), 307–317 (2013)
24. Chan, P.P.K., Song, W.K.: Static detection of android malware by using permissions and API calls. In: 2014 International Conference on Machine Learning and Cybernetics, vol. 1, pp. 82–87. IEEE (2014)
25. Wang, X., Wang, W., He, Y., Liu, J., Han, Z., Zhang, X.: Characterizing Android apps' behavior for effective detection of malapps at large scale. Future Gener. Comput. Syst. **75**, 30–45 (2017)
26. Fan, M., Liu, J., Wang, W., Li, H., Tian, Z., Liu, T.: Dapasa: detecting android piggybacked apps through sensitive subgraph analysis. IEEE Trans. Inf. Forensics Secur. **12**(8), 1772–1785 (2017)
27. Al Shorman, A., Faris, H., Aljarah, I.: Unsupervised intelligent system based on one class support vector machine and grey wolf optimization for iot botnet detection. J. Ambient Intell. Humaniz. Comput. **11**(7), 2809–2825 (2020). https://doi.org/10.1007/s12652-019-01387-y
28. Islam, T., Rahman, S.S.M.M., Hasan, M.A., Rahaman, A.S.M.M., Jabiullah, M.I.: Evaluation of N-gram based multi-layer approach to detect malware in android. Procedia Comput. Sci. **171**, 1074–1082 (2020)
29. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C.E.: Drebin: effective and explainable detection of android malware in your pocket. In: NDSS, vol. 14, pp. 23–26 (2014)
30. Yerima, S.Y., Sezer, S.: Droidfusion: a novel multilevel classifier fusion approach for android malware detection. IEEE Trans. Cybern. **49**(2), 453–466 (2018)
31. Kotsiantis, S., Kanellopoulos, D., Pintelas, P., et al.: Handling imbalanced datasets: a review. GESTS Int. Trans. Comput. Sci. and Eng. **30**(1), 25–36 (2006)
32. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: synthetic minority over-sampling technique. J. Artif. Intel. Res. **16**, 321–357 (2002)
33. Schapire, R.E.: Explaining adaBoost. In: Schölkopf, B., Luo, Z., Vovk, V. (eds.) Empirical Inference, pp. 37–52. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41136-6_5
34. Dorogush, A.V., Ershov, V., Gulin, A.: CatBoost: gradient boosting with categorical features support. *arXiv preprint* arXiv:1810.11363 (2018)
35. Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A.V., Gulin, A.: CatBoost: unbiased boosting with categorical features. *arXiv preprint* arXiv:1706.09516 (2017)
36. Chen, T., Guestrin, C.: XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery And Data Mining, pp. 785–794 (2016)
37. Nguyen, L.T.K., Chung, H.H., Tuliao, K.V., Lin, T.M.Y.: Using XGBoost and skip-gram model to predict online review popularity. SAGE Open **10**(4), 1–17 (2020). https://doi.org/10.1177/2158244020983316
38. Friedman, J.H.: Stochastic gradient boosting. Comput. Stat. Data Anal. **38**(4), 367–378 (2002)
39. Chawla, N.V.: Data mining for imbalanced datasets: an overview. In: Maimon, O., Rokach, L. (eds.) Data Mining and Knowledge Discovery Handbook, pp. 875–886. Springer, Boston (2009). https://doi.org/10.1007/978-0-387-09823-4_45
40. Zien, A., Kramer, N., Sonnenburg, S., Ratsch, G.: The feature importance ranking measure. In: Buntine, W., Grobelnik, M., Mladenić, D., Shawe-Taylor, J. (eds.) Machine Learning and Knowledge Discovery in Databases, vol. 5782, pp. 694–709. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04174-7_45