Malicious Software Detection For Android Devices Using Ensemble Learning

Samuel Kim
Department of Computer Science
Georgia State University
Atlanta, GA 30302

Abstract — **Increased use of permission-based security systems in mobile phone applications introduced concerns over privacy and security issues. Given that Android OS occupies much of the market share in the mobile application market, this poses an immediate security risk that must be addressed. User ignorance during the process of installing and usage of mobile apps makes unsuspecting users an easy target for attackers with malicious intent. Common methods to detect malicious behavior include using a list of enabled application permissions as a feature set for machine learning algorithms. Classification algorithms such as Support Vector Machines and Random Forest have shown to perform well in detecting malicious software. The research in this paper displays an analysis of classification results from various ensemble learning methods. Our results show that Adaptive boosting (AdaBoost) performed the best in terms of classification accuracy (91.65%) with AUC value of 95%.**

Keywords—android, malware detection, principal component analysis, ensemble learning

I. Introduction

As of April 2021, the Android Operating System currently holds 72.19% market share of mobile operating systems worldwide. Open-source software with popularity of this magnitude can comes with underlying security risks. Closed-sourced software such as iOS allow users to only install applications through their platform, mitigating risk of downloading malicious software. Android OS is an open-source platform allowing users to install applications from third-party platforms, providing convenient opportunities for attackers to distribute malware among many devices.

Malware in Android devices can be detected based on the permissions it requests from the user [1]. Android app permissions can be located and extracted by decompiling APKs (Android application packages) and extracting permission data from the AndroidManifest.xml file. Applications with malicious intent may request confidential data from unsuspecting users. Use of smartphones for financial transactions and other confidential information motivates attackers to find ways to access this data. This paper will aim to present an analysis of results from various machine learning algorithms in order to develop an efficient malware detection system on Android devices.

II. Related Work

Firdausi et al. analyzed the performance of several machine learning techniques used in behavior-based malware detection [2]. Best first search algorithm was used for feature selection and the authors discovered that performing feature selection dramatically improved performance on the Naïve Bayes classifier, increasing accuracy from 65.4% to 92.3%. Other classifiers include SVM, decision trees, and K-nearest neighbor all of which output 92% (+/- 0.9) accuracy.

Zhang et al. implemented a new semantics-based android malware detection method using Random Forest and static analysis and achieved highest accuracy of 89.6% [3]. Semantic-based features, which can resist code obfuscation, were extracted from APK files as well as app attributes and permissions.

A. Sangal et al. utilized Principal Component Analysis for feature selection and trained various machine learning models using Android permissions and intent as feature set for malware detection. Random Forest was determined the best classifier with 96.05% accuracy [4].

W. Kuo et al. proposed a malware detection system that which combines SVM or Random Forest algorithms with hybrid analysis to generate a. detection module to detect Android malware [5]. Average accuracy rate achieved 88% by using hybrid detection modules, with true positive rate reaching 89% accuracy [5].

III. Proposed Method

This section will describe the methodology behind my proposed method, and the different components involved. The classification algorithms I experimented with were the best performing classifiers used by Firdausi et al [2]. These algorithms include Support Vector Machine, Naïve Bayes, and Decision Tree classifiers.

A. Dataset

The data used in this study was taken from the CICInvesAndMal2019 dataset, released by the authors of the paper cited in *[6]*. The CICInvesAndMal2019 dataset consists of 5,000 captured malware samples (426 malware and 5,065 benign) found on Android devices. The malware samples are classified into 4 categories: Adware, Ransomware, Scareware, and SMS malware.

B. Feature selection

428 different types of Android permissions were used as feature labels in the dataset. The list of Android permissions used were downloaded from the GitHub repository in [7]. Figure 1 displays a portion of the permissions selected as features.

```
android.permission.STORAGE
android.permission.READ_EXTERNAL_STORAGE
android.permission.READ_CONTACTS
android.permission.WRITE_CONTACTS
android.permission.GET_ACCOUNTS
android.permission.ACCESS_FINE_LOCATION
android.permission.CAMERA
android.permission.USE_FINGERPRINT
android.permission.READ_CALENDAR
android.permission.WRITE_CALENDAR
android.permission.ACCESS_CORSE_LOCATION
android.permission.PREVENT_POWER_KEY
```
Fig 1. Android permissions

Principal Component Analysis was utilized as a feature selection technique as an experiment. However, I found that it decreased performance in terms of classification accuracy.

C. *Feature Extraction*

Out of the 5,000 captured malware samples from CICInvesAndMal2019, 433 samples were downloaded to extract permission data and store into a CSV file. Figure 2 displays the quantities of each sample.

Sample Type	**Number Of Samples**
Adware	20
Scareware	46
SMS Malware	39
Ransomware	21
Benign	305

Fig 2. Number of each malware sample

To prepare the dataset to train and test our ensemble learning model, I implemented a Python script to write to a CSV file containing our data. The program iterates through every sample and extracts permission names found from the sample's APK. Each column value for each sample is filled with 1 if permission is found, or 0 if not found.

D. *Data Pre-Processing*

The dataset does not contain any missing values or outliers due to the data taking the form of binary values (0 or 1). Other than removing empty rows, there was no need for further data cleaning.

E. *Ensemble Learning*

Ensemble learning merges the decisions of multiple classifiers into one, either by averaging the results or accepting majority vote. The idea behind this methodology is that a decision made in agreement from multiple sources may be more reliable than from a single source. Ensemble methods aim to reduce bias and/or variance of "weak learners" by combining several of them together to form one "strong learner" that achieves better performance.

3 different ensemble learning methods tested on dataset:

Voting Ensemble classifier

Adaptive Boosting (AdaBoost)

Bootstrap Aggregating (Bagging)

Voting ensemble classifier is an ensemble model that collects the classification results returned from each model and output the final prediction according to a majority vote (hard voting). Soft voting is used for regression problems that output quantitative responses by averaging the predicted values.

Adaptive Boosting (AdaBoost) is a boosting algorithm that trains weak learners sequentially where each model adapts and learns from the previous model and combines them to form a final prediction. Boosting methods aim to reduce bias while Bagging aims to reduce variance on the base models.

Bootstrap Aggregating (Bagging) trains weak learners independently from each other in parallel and aims to form an ensemble model that is more robust than the individual models themselves. These models can be trained in parallel and concurrently unlike Boosting, which trains base models sequentially in an iterative fashion.

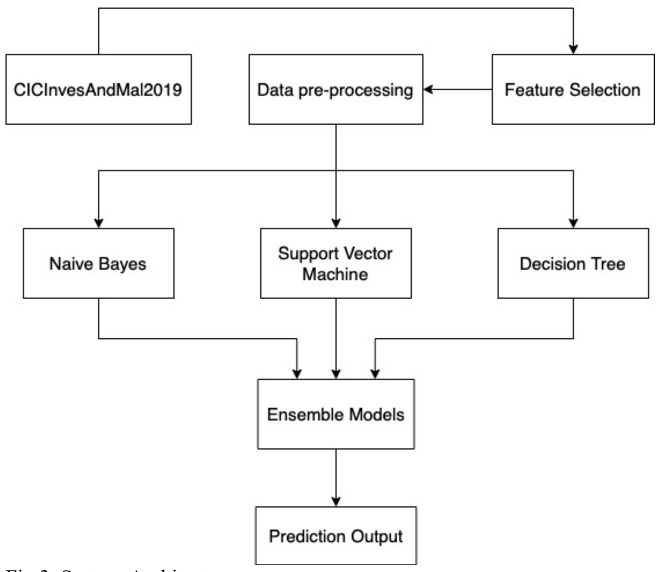


Fig 3. System Architecture

IV. Experimental Results

A. *Feature Reduction with PCA*

Principal Component Analysis (PCA) is a popular feature reduction technique used to reduce dimensionality of large datasets by transforming a large set of variables into linearly independent variables (eigen-vectors). These eigen-vectors, also known as principal components, can be used as new set of features to feed into machine learning models. PCA aims to overcome overfitting issues by removing noisy/irrelevant data.

The initial step before performing PCA is to determine the appropriate number of principal components to utilize. The general rule of thumb is to use the number of principal components needed to account for the largest possible amount of variance in the dataset.

Using the explained variance ratio, we can visualize the number of principal components needed to account for at least 95% variance. Figure 4 shows we will need to use roughly 45-50 principal components to account for at least 95% variance for our dataset.

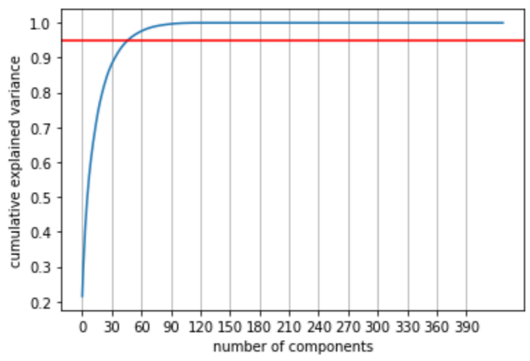


Fig 4. Number of Principal Components needed with respect to amount of variance accounted for

After performing PCA on the dataset and fitting each model with the generated principal components as features, I found that each model performed worse in all aspects. I am led to believe this is due to the dataset not being large enough. Important features may have been removed that were needed for better classification, resulting in reduced accuracy.

B. *Performance Metrics*

The following chart displays performance evaluation metrics for machine learning models SVM, Naïve Bayes, and Decision Tree. Where k=5, K-fold cross validation was used to split the dataset into multiple folds to minimize overfitting.

	Accuracy	**Precision**	**Recall**	**F1-Score**
Decision Tree	78.93%	96.15%	71.43%	81.97%
Naïve Bayes	78.93%	93.33%	40%	55.99%
SVM	90.01%	89.29%	71.43%	79.37%

Fig 5. Performance evaluation metrics

While precision indicates how many predicted positives were identified correctly out of all positives, recall indicates how many predicted positives were identified correctly out of all the positive predictions that could have been made. Both recall and precision measure how accurately a model can correctly identify the data.

The formula for computing recall is as follows, where TP = True Positive and FN.= False Negative (wrongly predicted positive):

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}}$$

F1-Score is defined as the harmonic mean of precision and recall and is needed when we wish to seek a balance between the two evaluation metrics. As shown in Figure 5, Naïve Bayes appears to be a much weaker model compared to SVM and decision tree.

$$\text{F1-Score} = \quad 2 * \frac{\text{precision}*\text{recall}}{\text{precision+recall}}$$

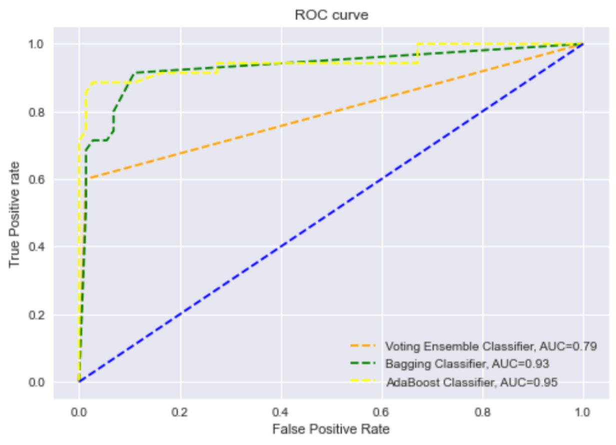
Now we will examine how these metrics improve when using these classifiers as base models for our chosen ensemble learning methods.

The voting classifier is composed of all 3 models evaluated previously (SVM, decision tree, Naïve Bayes) and performed the worst out of the three ensemble methods. Bagging classifier uses SVM as the base model estimator, performing second best. The AdaBoost classifier uses decision trees as its base model estimator, ranked as the top performer of all tested ensemble learning methods.

	Accuracy	**Precision**	**Recall**	**F1-Score**
Voting Classifier	89.77%	95.45%	60%	73.68%
Bagging	90.70%	92.59%	71.43%	80.65%
AdaBoost	91.65%	96.77%	85.71%	90.90%

C. *Evaluating Model Performance*

Visualizing the AUC (Area Under the Curve) helps us determine the performance of binary classification models. It serves as a great measurement of model performance. AUC values range from 0 to 1, where values closer to 1, indicate better performance, and values closer to 0 indicate worse performance.



The AdaBoost classification model can be determined as the best performing ensemble model with the highest AUC value of 0.95. Using this performance metric allows us to decide on the best model to use for future predictions.

V. Conclusion

An operating system like Android that is used by 70% of the worldwide smartphone market poses an immediate security risk that requires consistent research and development. We proposed a malicious software detection system using the AdaBoost ensemble learning algorithm with results that validate our idea that combining multiple "weak learners" to create one "strong learner" can improve classification performance regarding malware detection. In the future, I hope to further implement this model to analyze APK files directly from Android Devices and detect upon installation of applications.

References

[1] P. R. K. Varma, K. P. Raj and K. S. Raju, "Android mobile security by detecting and classification of malware based on permissions using machine learning algorithms," in *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, Palladam, India, 2017.

[2] I. Firdausi, C. lim, A. Erwin and A. S. Nugroho, "Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection," in *Second International Conference on Advances in Computing, Control, and Telecommunication Technologies*, Jakarta, Indonesia, 2010.

[3] X. Zhang and Z. Jin, "A new semantics-based android malware detection," in *2nd IEEE International Conference on Computer and Communications (ICCC)*, Chengdu, China, 2016.

[4] A. Sangal and H. K. Verma, "A Static Feature Selection-based Android Malware Detection Using Machine Learning Techniques," in *2020 International Conference on Smart Electronics and Communication (ICOSEC)*, Trichy, India, 2020.

[5] W.-C. Kuo, T.-P. Liu and C.-C. Wang, "Study on Android Hybrid Malware Detection Based on Machine Learning," in *2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS)*, Singapore, 2019.

[6] L. Taheri, A. F. A. Kadir and A. H. Lashkari, "Extensible Android Malware Detection and Family Classification Using Network-Flows and API-Calls," in *International Carnahan Conference on Security Technology (ICCST)*, Chennai, India, 2019.

[7] Sivadas, "GitHub," 6 Dec 2020. [Online]. Available: https://github.com/anoopmsivadas/android-malware-detection.

[8] S. Zhernakov and G. N. Gavrilov, "Malicious software detection in operating system (OS) for mobile devices (the case of Android OS)," in *2016 13th International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE)*, Novosibirsk, Russia, 2016.

[9] S. Somasundaram, D. Kasthurirathna and L. Rupasinghe, "Mobile-based Malware Detection and Classification using Ensemble Artificial Intelligence," in *International Conference on Advancements in Computing (ICAC)*, Malabe, Sri Lanka, 2019.