Identification of malicious android app using manifest and opcode features

Article · May 2016

DOI: 10.1007/s11416-016-0277-z

CITATIONS
10

READS
230

3 authors, including:

P. Vinod
SCMS School of Engineering & Technology, Ernakulam, Kerala
52 PUBLICATIONS **258** CITATIONS

SEE PROFILE

Identification of malicious android app using manifest and opcode features

M. V. Varsha, P. Vinod & K. A. Dhanya

Journal of Computer Virology and Hacking Techniques

ISSN 2274-2042

J Comput Virol Hack Tech
DOI 10.1007/s11416-016-0277-z

Springer

J Comput Virol Hack Tech
DOI 10.1007/s11416-016-0277-z

ORIGINAL PAPER

Identification of malicious android app using manifest and opcode features

M. V. Varsha[1] · **P. Vinod**[1] · **K. A. Dhanya**[2]

Received: 8 November 2015 / Accepted: 25 April 2016

Abstract In this paper, we propose a statistical approach for smartphone malware detection. A set of features such as hardware, permission, application components, filtered intents, opcodes and strings are extracted from the samples to form a vector space model. Feature selection methods such as Entropy based Category Coverage Difference (ECCD) and Weighted Mutual Information (WI) are used to choose the prominent features. The performance of the system is analyzed using classifiers, Random Forest, Rotation Forest and Support Vector Machine (SVM). The system was evaluated on individual models as well as Meta feature space model for both malware and benign features. It was observed that the meta feature space model with malware features provide the best results for both feature selection. For ECCD, Random Forest classifier performs better [Dataset 1—0.972, Dataset 2—0.976 and Dataset 3—0.969] whereas in the case of WI, SVM gives highest F-measure [Dataset 1—0.993, Dataset 2—0.994 and Dataset 3—0.992]. From the overall analysis on the system, we conclude that the malware model outperforms it's benign counterpart and also that WI is a better feature selection technique compared to ECCD.

Keywords Android malware · Manifest · Opcode · Strings · Feature selection · Meta feature · Classifier

✉ P. Vinod
pvinod21@gmail.com

[1] Department of Computer Science and Engineering, SCMS School of Engineering and Technology, Ernakulam, Kerala, India

[2] Centre for Cyber Security, Amrita School of Engineering, Amrita Vishwa Vidyapeetham University, Coimbatore, Tamilnadu, India

1 Introduction

Android is the most popular platform developed by Google. It is based on a Linux kernel. Since smartphones exhibit similar capabilities compared to personal computer systems, the demand of smartphone devices have shown increased acceptance [5]. The popularity of smartphones thus attracted many organizations/companies to develop applications (app) for such equipments. The soaring demand for Android also attracted several attackers to target smart devices installed with Android OS. Cyber crimes started revolving around the development of malicious applications targeting legitimate apps which have maximum number of downloads and high content rating. Sending premium messages, making phone calls to premium numbers that users are unaware about and stealing user data are some of the reported malicious activities.

According to the report from Gartner [5], there is a 4.3 % rise in the worldwide tablet sales in 2014 and is expected to reach 237 million units in 2015. The global combined shipment of PCs, ultramobiles, tablets and mobile phones has shown a rise of 2.8 % (2.5 billion units) in 2015 over 2014. However, the global PC market experienced a 2.4 % decline from 2014, i.e., a total of 306 million units in 2015.

A common method used for identifying malware is the extraction of patterns from each suspicious file matching with the signature repository. The primary limitation of signature based detector is its inefficiency in the identification of zero day vulnerability. Also, obfuscated files thwart signature based approach. However, suspicious specimens can be effectively separated out from devices using machine learning approach. The improved performance of scanners can be ascertained by obtaining significant features/attributes that represent the behavior of a class. Moreover, the selection of robust classification algorithm subjected with enough pat-

Published online: 14 May 2016

terns can minimize the false positive rate which is usually a remarkable property of most of the commercial AV products. A review conducted on the research works published between 2010 and 2014 in the domain of mobile malware detection by authors in [18] concluded that out of 100 articles only 8 research works considered feature selection techniques for malware detection. It is also reported that features are extracted equally using static and dynamic approaches. However, for improved performance elimination of irrelevant attributes can result in lesser classifier misclassification. Therefore our study is focussed on the use of feature selection methods for developing optimal feature vectors to achieve better detection.

The contributions of this study are follows:

- The evaluation was performed on malware samples from Drebin project [10], it expected to have inexhaustive malware applications from 179 malicious families.
- The empirical validation of independent classification models developed by synthesized features, extracted on employing feature selection methods with that of models formed by composite feature space is conducted.
- To validate the effect of variable feature length and different classification algorithms in the performance of statistical malware detection techniques.

The remainder of the paper is organized as follows: Sect. 2 address the related work in the area of smart phone malware detection. Section 3 thoroughly explains the proposed methodology. The experimental setup is described in Sect. 4 and the results of the experiments are analyzed in Sect. 5. Section 6 presents the inference of the experiments and the conclusion of the work is covered in Sect. 7.

2 Related works

A lightweight method to detect malicious applications in mobile devices was proposed in [10], the authors found that it identifies Android malware with a high accuracy. With the help of a set of predefined security rules, authors in [17] developed Kirin which performs the following tasks: It certifies an application during installation time, analyzes different types of malicious behaviors and defines rules associated with permission and intent information. In [20], authors analyzed the real-world mobile malware between 2009 and 2011 and also discussed the importance of usage of permission distribution to classify an application as benign or as malware. For classification, a hierarchical mixture model was developed in [26] by applying an approach called simple Naïve Bayes to requested permissions. A comparison was done on the permissions distributed between malware and benign applications [29]. As a result the critical permissions for detecting malware was determined. Authors in [34] introduced an approach in which the permissions are systematically ranked with respect to their risks to Android system.

An automated analyzing system called Mobile-Sandbox was proposed in [32] where an application's permission and intent information are parsed to detect the suspicious activities among them, followed by performing dynamic analysis to log actions (mainly those based on native API calls). In [31], a framework to examine Android devices was proposed which uses supervised anomaly detection techniques in order to identify suspicious and abnormal activities in mobile devices. A static analysis system for detection and classification of Android malware was introduced by authors in [23] using creator's information as a feature. In [24], a method was developed to extract the main executable code from memory by altering a part of Dalvik Virtual Machine. A systematic study on the existing Android malware is described in [36]. An approach capable of distinguishing between the benign and malicious applications bearing similar names and of the same version was proposed in [13] to analyze the anomalous behaviour of Android applications.

Authors in [21] presented a semantic based static analysis called Apposcopy, for detecting the malware that steals user's private information. A robust approach called DroidAPIMiner [9] was developed with the help of machine learning algorithms to automatically classify the malicious applications within a malware family. An effective Mobile Application Security Triage (MAST) architecture was proposed [15], which uses the Multiple Correspondence Analysis (MCA) a statistical method to solve the manual and resource-intensive automated analysis at market-scale. Authors in [27] introduced DroidChameleon, a systematic framework which includes various transformation techniques. It is used for evaluating the existing anti-malware software for Android and analyzing their resistance against the common obfuscation techniques. A static analysis system called DroidMat [35] was developed to detect malicious apps by extracting features like permissions, Intent message passing, deployment of components and API calls, which characterizes the behavior of Android applications. A framework to detect Android malware applications was proposed by authors in [11]. It includes developing a malware detection system on Android based on machine learning to identify malicious applications and enhance the security and privacy of smartphone users. In [19], a static analysis tool called Stowaway was introduced, which explores whether more permissions are requested by an application than that it actually uses with the help of a permission-to-API-calls map. In our previous work [33], a static method to classify Android malware based on machine learning using features extracted from manifest and dex files was reported. An F-measure of 0.976 (accuracy 98.14 %) was achieved with heterogeneous feature space (malware features) using Entropy based

Category Coverage Difference feature selection for Random Forest classifier.

In [37] authors propose a novel approach for identifying unseen malware samples by constructing Weighted Contextual API dependency graph. This weighted graph was statically extracted from each apk sample. The context of samples were preserved by filtering out API names, entry point and the constants parameters used by each API. Further a graph database was developed from the training samples, which were mapped to a vector space model. Subsequently new samples (files in test) were predicted with 93 % accuracy using priorly build Naive Bayes learning model. Later anomaly based detector was proposed which resulted in 2 % FNR and 5.1 % FPR respectively.

3 Proposed methodology

The framework of proposed methodology is depicted in Fig. 1. A subset of the benchmark dataset called Drebin provided by Arp et al. [10] is used for evaluating the proposed method. It is discussed in detail in the following subsections.

3.1 Dataset description

Drebin Dataset includes malware samples collected from the Android Malware Genome Project [2], Google Play store [6], Chinese [38] and Russian markets, Android app websites [39,41], Malware forums and Security blogs [40]. The apps downloaded were submitted to the VirusTotal service to distinguish the app as benign or malware. Drebin [4] consists of six folders named drebin-0 to drebin-5, containing 5560 malware samples organized as follows: drebin-0 to drebin-4 consisting of 1000 malware apks and drebin-5 with 560 apk samples. Out of six folders in Drebin dataset, only three folders (Drebin-3, Drebin-4 and Drebin-5) are used for investigation. A total of 1631 benign apk files were downloaded from Google Play store for analysis. Three datasets were formed for the experiment which consisted of Dataset 1 (Drebin-3 and benign), Dataset 2 (Drebin-4 and benign) and Dataset 3 (Drebin-5 and benign).

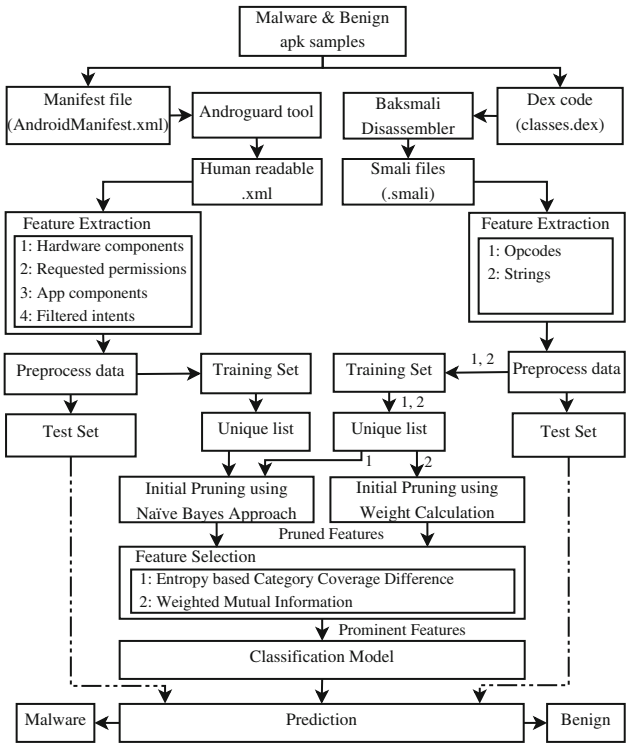

Fig. 1 Architecture diagram of proposed methodology

3.2 Preprocess samples

The input to the proposed system is the benign and malware .apk samples. The Android apps [3] are deployed in Dalvik bytecode. It is portable and appear in the form of .dex format. Tools like Baksmali [7] and Androguard [1] are used to disassemble the .apk samples. With the help of Baksmali, the Dalvik executable (.dex) is converted into readable Dalvik byte code (.smali). Androguard tool is used to convert the Android manifest file into human readable .xml format. Figure 2 shows the set of features extracted and used in our study. Features such as hardware, permission, filtered intents and application components are extracted from each .xml file. Opcodes and strings are extracted from the smali files.

Training and test sets are formed by randomly dividing the benign and malware .apk samples in each dataset (Dataset 1, Dataset 2 and Dataset 3) in the 60:40 ratio. If the number of samples in the training set is less, then the parameter/attribute used for estimation will have high variance. Likewise, if the test samples are fewer, then the performance metrics will indicate high variance. While developing a classification model we should avoid both conditions. If the number

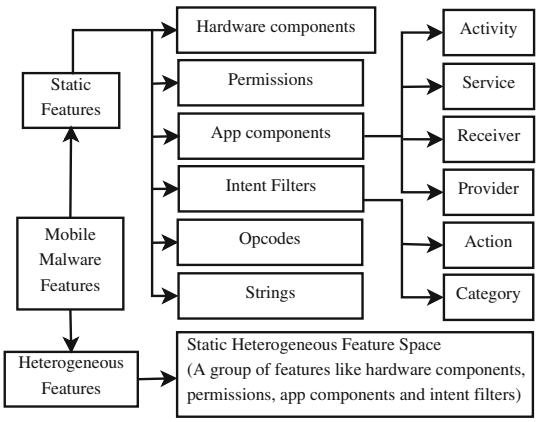

Fig. 2 Application features

Table 1 Unique list for permission

Feature	df_B	df_M
android.permission.access_network_state	614	365
android.permission.access_wifi_state	223	226
android.permission.call_phone	125	87
android.permission.get_accounts	188	47
android.permission.internet	732	554
android.permission.receive_boot_completed	193	255

Table 2 Naïve bayes score [Benign (B_score) and Malware (M_score)] for each feature in the unique list (permission)

Feature	B_score	M_score
android.permission. access_network_state	0.627171	0.372829
android.permission. access_wifi_state	0.496659	0.503341
android.permission. call_phone	0.589623	0.410377
android.permission. get_accounts	0.8	0.2
android.permission. internet	0.569207	0.430793
android.permission. receive_boot_completed	0.430804	0.569196

of samples are fewer say 100s, then k-fold cross validation is preferable. Here, $(k-1)$ instances are used during the training phase and the remaining subset is used as test set. This experiment is repeated k times and average accuracy is used to estimate the quality of classification model. However, if the number of samples are large (eg: 1000 instances) then data set can be divided into train and test set, where 60–80 % of samples are utilized for generating models and the remaining 40–20 % are used as test samples. The basis is that the models developed with fewer instances cannot represent the entire population of the data set. Another alternative approach that could be used is to create the classification model with 60 % of instances, then validate the model using 20 % of samples and finally test using remaining 20 % of specimens.

3.3 Elimination of rare attributes

Naïve bayes approach is used to perform initial pruning of attributes extracted from manifest and .smali files. A unique list [Table 1] consisting of distinct features from all samples, document occurrence number in benign (df_B), term occurrence number in benign (tf_B), document occurrence number in malware (df_M) and term occurrence number in malware (tf_M) are estimated. df_B denotes the number of benign samples containing the feature and tf_B is the no. of times the feature occurs in all benign samples. Similarly, df_M is the number of malware samples containing the feature and tf_M denotes the number of times the feature occurs in all malware samples.

Table 3 Percentage of benign (B) and malware (M) features obtained after initial pruning for all feature sets in three datasets

Features	Dataset 1		Dataset 2		Dataset 3	
	B	M	B	M	B	M
Hardware	97.2	41.7	97.2	44.4	97.2	27.8
Permission	94.2	24.5	95.1	22.9	93.4	23.4
Action	92.5	9.8	93.8	8.4	94.4	7.6
Category	94.4	10.2	94.4	8.8	94	10.1
Activity	89.6	11.1	90	10.9	91.3	9.4
Service	85.5	15.3	83.9	16.9	89.6	11.1
Receiver	85	16.2	83.8	17.4	88.6	12.4
Provider	97.7	2.6	97.9	2.6	98.5	2.1
Opcodes	60.2	60.2	60.2	60.2	60.2	60.2
Strings	72.4	64.9	73.3	64	70.8	66.5

For each feature in the unique list, the Naïve Bayes score is determined using Eq. 1.

$$P(C_i|f) = \frac{P(f|C_i)P(C_i)}{P(f)} \tag{1}$$

where f denotes the feature and C_i represents the class (Benign (B) or Malware (M)). $P(f|C_i)$ is the conditional probability of a feature f, given the class C_i to which it belongs to. $P(C_i)$ and $P(f)$ are the marginal probabilities of the class C_i and feature f.

The features are then sorted in decreasing order based on the Naïve Bayes score obtained for benign and malware. Finally, the relevant features with score greater than 0 are extracted from the sorted list, resulting in two pruned outputs, one with benign and the other with malware features. Similar steps are followed for all the feature sets except strings. The Naïve Bayes score obtained for permission feature set is shown in Table 2.

In the case of strings initial pruning is carried out by calculating weight and setting a threshold such that features with weight $\geq$ threshold are extracted. Calculate weight (w) for each feature a_i in benign using Eqs. 2, 3 and 4.

$$w(a_i) = x * y \tag{2}$$

$$x = \frac{tf_{a_i}}{\sum_{a_i} tf_{a_i}} \tag{3}$$

$$y = \frac{df_{a_i}}{N} \tag{4}$$

where tf_{a_i} is the term occurrence number of a feature a_i, df_{a_i}: document occurrence number of a feature a_i and N is the total

Table 4 Total number of features for all attribute sets in three datasets

Features	Dataset 1	Dataset 2	Dataset 3
Hardware	36	36	36
Permission	556	551	561
Action	2763	2725	2707
Category	216	216	217
Activity	13243	13190	13002
Service	1793	1827	1711
Receiver	1994	2022	1913
Provider	342	341	339
Opcodes	211	211	211
Strings	228	225	233

number of documents. All the features are sorted in decreasing order of weight w. Then the features with $w \geq 0.0001$ are extracted separately for benign and malware features. Table 3 describes the number of features obtained after initial pruning for all feature sets (Table 4).

3.4 Feature selection

Feature selection is the process of extracting the prominent attributes by discarding redundant and irrelevant features from the feature set. It aims to reduce the dimensionality of the feature space without reducing the performance of the classifier. It is capable of significantly decreasing the time required for model creation and prediction. In order to extract the significant features, Entropy based Category Coverage Difference (ECCD) and Weighted Mutual Information (WI) are the feature selection methods that is been used. The proposed feature selection methods are briefly explained in following subsections:

3.4.1 Entropy based category coverage difference

Entropy based Category Coverage Difference (ECCD) [25] depends on the distribution of the documents involving the feature in the class and exploits the entropy of the feature. The presence of a feature in only one class will result in minimum entropy (equal to 0), indicating that the feature might be a good discriminant feature for classification. The entropy will be maximum, if the feature appears in all the classes with same frequency. The steps to be followed for determining ECCD is described in Algorithm 1. After calculating ECCD, the features are sorted in decreasing order of the score and the significant features having score greater than 0 are selected for generating the model. Variable feature length in steps of 10 attributes are considered. The procedure is repeated for all attributes in the training set. The Table 5 shows an example of ECCD score for malware samples (training set). The number of extracted attributes (both benign and malware) for all feature categories on the application of ECCD feature selection is shown in Table 6.

Algorithm 1 Entropy based Category Coverage Difference

Input:

- F; F is the set of features obtained after initial pruning, where k represents feature ($k \in F$) and D is the number of pruned features ($D < N$) [N: Total number of features]
- $C \leftarrow \{M, B\}$; C is the target class, where M denotes Malware and B is Benign

Output

- F': Set of significant features, where $|F'| \leftarrow n$ such that $1 \leq n < D$

$F' \leftarrow \phi$ ▷ F' initialised to null
$E_{max} \leftarrow 0$ ▷ E_{max} initialised to zero
for each feature $k \in F$ **do**
  $P_{k|M} \leftarrow df_{k,M}/|M|$ ▷ $P_{k|M}$ denotes the conditional probability of feature k given the class (M) to which the feature belongs, $df_{k,M}$ is the document occurrence number of feature k in malware samples and $|M|$ denotes the number of malware files in training set
  $P_{k|B} \leftarrow df_{k,B}/|B|$ ▷ $P_{k|B}$ denotes conditional probability of feature k given the class (B) to which feature belongs, $df_{k,B}$ is the document occurrence number of feature k in benign samples and $|B|$ denotes number of benign files in training set
  $z_k^M \leftarrow df_{k,M}/(df_{k,M} + df_{k,B})$ ▷ z_k^M is the occurrence of a feature k in class M
  $z_k^B \leftarrow df_{k,B}/(df_{k,M} + df_{k,B})$ ▷ z_k^B is the occurrence of a feature k in class B
  $E_k \leftarrow -[z_k^M log(z_k^M) + z_k^B log(z_k^B)]$ ▷ E_k is the entropy of feature k
  if $E_k > E_{max}$ **then** ▷ Determine the maximum entropy value E_{max}
    $E_{max} \leftarrow E_k$
  end if
end for
for each feature $k \in F$ **do**
  $ECCD_k \leftarrow (P_{k|M} - P_{k|B})[(E_{max} - E_k)/E_{max}]$
  if $ECCD_k > 0$ **then** ▷ Discard feature k if score ≤ 0
    $F' \leftarrow F' \cup k$ ▷ Add feature k to the set of significant features
  end if
end for
sort(F') ▷ Sort features in decreasing order of $ECCD$ score
return F'

3.4.2 Weighted mutual information

Mutual Information (MI) is a theoretical measure often used to evaluate features. It measures the amount of information obtained from the value of a feature (e.g. presence or absence of a feature) belonging to a document and also the class to which the document belongs to. In other words, it measures the dependence between two random variables. In the context of feature selection, the dependence between a feature and the

Table 5 ECCD score for permission (malware)

Feature	ECCD score
android.permission.read_phone_state	0.197185
android.permission.receive_sms	0.095134
android.permission.send_sms	0.127839
android.permission.delete_packages	0.004790
android.permission.internet	0.054617
android.permission.access_wifi_state	0.058401

Table 6 Number of extracted benign (B) and malware (M) features for all feature sets in three datasets after applying ECCD

Features	Dataset 1		Dataset 2		Dataset 3	
	B	M	B	M	B	M
Hardware	13	1	14	1	7	2
Permission	29	75	31	68	29	65
Action	47	16	43	17	37	19
Category	6	4	5	2	5	4
Activity	25	78	39	78	21	70
Service	2	12	2	12	1	11
Receiver	4	19	5	19	2	17
Provider	0	1	1	1	0	2
Opcodes	124	20	124	16	124	9
Strings	80	5	76	8	79	8

target (e.g. the class labels) is measured. Here the weighted variant of the Mutual Information called Weighted Mutual Information (WI) [30] is applied. Algorithm 2 describes the method to calculate WI.

A weight $w \geq 0$ is imposed on each feature along with the combination of input and target values. This gives relevance to specific features. This is used to weight the influence of different input features. If WI is zero, both variables are independent and contain no information about each other, thus the feature is irrelevant for the target. Higher WI values indicate more information about the target and hence a higher relevance.

WI is computed for each attribute in different feature categories. The features are then sorted in descending order based on WI score. Next the prominent features from the sorted list is extracted to generate the model. Feature length in intervals of 10 (10, 20, . . . , 100), 50(150, 200, . . . , 2500) and 500 (3000, 3500, etc.) until full feature space is considered. This procedure is repeated for all attributes in different feature category belonging to malware and benign training set. An example of WI scores for permissions requested by malware apps are shown in Table 7.

Algorithm 2 Weighted Mutual Information

Input:

- F_M; F_M is the malware attribute set obtained after pruning, where i represents the attribute ($i \in F_M$) and D_M is the number of pruned malware attributes ($D_M < N$) [N: Total number of attributes]
- F_B; F_B is the benign attribute set obtained after pruning, where i represents the attribute ($i \in F_B$) and D_B is the number of pruned benign attributes ($D_B < N$)
- $C \leftarrow \{M, B\}$; C denotes the target class, where M is Malware and B is Benign

Output

- $F_M^{'}$: Set of prominent malware attributes, where $|F_M^{'}| \leftarrow d_M$ such that $1 \leq d_M \leq D_M$
- $F_B^{'}$: Set of prominent benign attributes, where $|F_B^{'}| \leftarrow d_B$ such that $1 \leq d_B \leq D_B$

$F_M^{'} \leftarrow \phi$ ▷ $F_M^{'}$ initialised to null
$F_B^{'} \leftarrow \phi$ ▷ $F_B^{'}$ initialised to null
$tf_{Msum} \leftarrow 0$ ▷ tf_{Msum} initialised to zero
$tf_{Bsum} \leftarrow 0$ ▷ tf_{Bsum} initialised to zero
$P_M \leftarrow |M|/|C|$ ▷ P_M is the marginal probability of class M, $|M|$ is the number of malware samples in train set and $|C|$ is the total number of samples in train set
$P_B \leftarrow |B|/|C|$ ▷ P_B is the marginal probability of class B and $|B|$ is the number of benign samples in train set
for each feature $i \in F_M$ **do**
    $tf_{Msum} \leftarrow tf_{Msum} + tf_{i,M}$ ▷ tf_{Msum} is the sum of term occurrence numbers of all malware attributes and $tf_{i,M}$ is the term occurrence number of an attribute i in class M
end for
for each feature $i \in F_B$ **do**
    $tf_{Bsum} \leftarrow tf_{Bsum} + tf_{i,B}$ ▷ tf_{Bsum} is the sum of term occurrence numbers of all benign attributes and $tf_{i,B}$ denotes the term occurrence number of an attribute i in class B
end for
for each feature $i \in F_M$ **do**
    $x_i \leftarrow tf_{i,M}/tf_{Msum}$
    $y_i \leftarrow df_{i,M}/|M|$ ▷ $df_{i,M}$ is the document occurrence number of an attribute i in class M
    $w_i \leftarrow x_i * y_i$ ▷ w_i is the weight of malware attribute i
    $P_{i,M} \leftarrow df_{i,M}/|M|$ ▷ $P_{i,M}$ is the probability of attribute i given the class label M
    $P_i \leftarrow (df_{i,M} + df_{i,B})/|C|$ ▷ P_i is the marginal probability of an attribute i and $df_{i,B}$ is the document occurrence number of attribute i belonging to class B
    $WI_i \leftarrow w_i * P_{i,M} log[(P_{i,M} + 1)/(P_i * P_M)]$
    $F_M^{'} \leftarrow F_M^{'} \cup i$ ▷ Add attribute i to set $F_M^{'}$
end for
for each feature $i \in F_B$ **do**
    $x_i \leftarrow tf_{i,B}/tf_{Bsum}$
    $y_i \leftarrow df_{i,B}/|B|$
    $w_i \leftarrow x_i * y_i$ ▷ w_i is the weight of benign attribute i
    $P_{i,B} \leftarrow df_{i,B}/|B|$ ▷ $P_{i,B}$ is the probability of attribute i given the class label B
    $P_i \leftarrow (df_{i,M} + df_{i,B})/|C|$ ▷ P_i is the marginal probability of benign attribute i
    $WI_i \leftarrow w_i * P_{i,B} log[(P_{i,B} + 1)/(P_i * P_B)]$
    $F_B^{'} \leftarrow F_B^{'} \cup i$ ▷ Add attribute i to set $F_B^{'}$
end for
sort($F_M^{'}$) ▷ Sort malware attributes in decreasing order of WI
sort($F_B^{'}$) ▷ Sort benign attributes in decreasing order of WI
return $F_M^{'}$, $F_B^{'}$

Table 7 WI score for permission (malware)

Feature	WI score
android.permission.read_phone_state	0.056179
android.permission.receive_sms	0.005408
android.permission.send_sms	0.016122
android.permission.delete_packages	0.000013
android.permission.internet	0.063179
android.permission.access_wifi_state	0.005658

3.5 Model generation

Training models are generated at variable feature lengths by using the features with high scores (extracted post feature selection). For each feature set (such as hardware, permission, action, category, activity, service, receiver, provider, opcodes and strings) independent models are constructed using benign and malware training files. The independent models are developed for pruned features obtained with both the feature selection methods used here viz. ECCD and WI. A boolean Vector Space Model (VSM) is generated for attributes/features extracted from manifest files. In this representation of vector (for each sample) 1/0 denote the presence/absence of feature in an instance. In case of opcodes and strings, files are also represented as vectors along with frequencies of attributes (number of times a feature occurs in the sample). Learning models are developed using classification algorithms (Support vVector Machine, Rotation Forest and Random Forest) implemented in WEKA [8]. Motivated by the performance and the range of applicability of these classification algorithm in malware detection as in [12,16,28], we generated the models using similar algorithms. The model with maximum Area under Receiver Operating Characteristic curve (AUC) at smaller feature length is selected as the best model for predicting new instances. The following subsections introduces the classification algorithms:

3.5.1 Random forest

Random Forest [12] is an ensemble learning method for classification, which contains a number of decision trees. Class labels are predicted by finding the mode of the classes' output from the individual trees. In this method, a group of decision trees are constructed with controlled variation using the idea of bagging with random selection of features. Random Forest is one of the most accurate classification algorithms. It is quite efficient in handling large data and can easily interpret the models created. It also estimates the feature variables' importance in classification and provides a method for finding missing data. Accuracy is maintained even if a large proportion of data is missing. The main disadvantage of Random Forest is the decrease in the speed of the algorithm with increase in the number of trees, for real-time prediction.

3.5.2 Rotation forest

Rotation Forest [28] is an approach for generating classifier ensembles with the help of independently trained decision trees. The individual decision trees are formed using Principal Component analysis (PCA) with features selected on random basis. The main aim of Rotation Forest is to build accurate and diverse individual classifiers. Diversity is achieved by rotating the feature space. Each decision tree uses a different set of axes. Accuracy is sought by performing sparse rotation, preserving all principal components in the rotated feature space and using whole dataset for training each classifier (with different extracted features).

3.5.3 Support vector machine (SVM)

The algorithm [16] analyzes the data and maps them into a multidimensional space. It is capable of effectively handling large dimensional input data. SVM identifies a hyper plane that separates the instances of classes in the multidimensional space. New instances are evaluated by mapping them to this region and observing to which side of the hyper plane do they lie. Even though SVM produces high accuracy, it is computationally expensive due to the implementation of quadratic programming.

3.6 Ensemble attribute space

The performance of the training models (obtained with each attributes in manifest files) are evaluated using metrics discussed in Sect. 4.1. The models with optimal feature length and higher values of AUC are used to create Meta feature space. This feature space is the aggregation/ensemble of the optimal features obtained for independent learning models. Let us assume a set E to denote the meta feature space then it would be represented as $E = \{H, P, A, C, A_t, S, R, P_r\}$ where:

- H is the hardware feature with dimensionality $h \ll D(H)$
- P is permission with feature length $p \ll D(P)$
- A is the action attribute with size $a \ll D(A)$
- C is category with attribute length $c \ll D(C)$
- A_t is activity with dimensionality $a_t \ll D(A_t)$
- S is service with feature length $s \ll D(S)$
- R is the receiver feature with length $r \ll D(R)$
- P_r is provider attribute with dimensionality $p_r \ll D(P_r)$

Here the functions $D(H), D(P), D(A), D(C), D(A_t), D(S), D(R)$ and $D(P_r)$ denotes the original dimensionality of the attributes.

Meta features were intended to be prepared for introducing maximum variability in the VSM of the class which might not have happened considering individual attributes. This ensemble model hence has finer representation of the behavior of instances.

3.7 Prediction

Unseen samples in the test set are predicted with two training models, one created with the individual attributes and the other ensemble or composite features.

4 Experimental setup

The experiments were performed on a Ubuntu 14.04 platform, Intel Core 3 and 4 GB of RAM. Malware samples from the Drebin dataset [4] and benign samples from the Google Play store [6] were used for conducting the experiments. Benign samples were submitted to VirusTotal in order to check for infection as infected files would hinder the generation of learning models. Thus the benign along with malware samples in each Drebin folder (Drebin-3, Drebin-4 and Drebin-5) were treated separately for evaluation, resulting in three datasets. i.e., Dataset 1 with 954 malware samples of Drebin-3 and 1464 benign files. The Dataset 2 consists of 949 malware files of Drebin-4 and 1464 benign samples. Dataset 3 includes 525 malware samples of Drebin-5 and 1464 benign files.

4.1 Evaluation parameters

For a detection system, the *F-measure* and the *AUC* (*Area Under ROC Curve*) act as important parameters to identify the robustness of the learned models. *F-measure* is defined as the harmonic mean of *precision* and *recall* and is determined using the Eq. 5. *Precision* and *recall* are calculated using Eqs. 6 and 7 respectively. The detection rate (*TPR*) (Eq. 7) and false positive rate (*FPR*) (Eq. 8) are also determined for classifying instances.

$$F\text{-}measure = \frac{2 * Precision * Recall}{Precision + Recall} \tag{5}$$

$$Precision = \frac{TP}{TP + FP} \tag{6}$$

$$Recall = TPR = \frac{TP}{TP + FN} \tag{7}$$

$$FPR = \frac{FP}{TN + FP} \tag{8}$$

False Positive (FP) denotes the number of benign misclassified as malware. When malware is correctly classified, it is considered as True Positive (TP). False Negative (FN) is the number of malware misclassified as benign. True Negative (TN) is the number of benign correctly classified as benign. Larger the area under the curve in the ROC, better is the performance of the classification system.

5 Results and discussions

The outcomes for Dataset 1 with manifest features is discussed in Sects. 5.1 and 5.2, the results for Dataset 2 and Dataset 3 is projected in Appendix.

5.1 Analysis with ECCD manifest features

The performance with manifest features extracted using Entropy based Category Coverage Difference (Sect. 3.4.1) for both malware and benign features is explained in subsequent paragraphs.

Table 8 depict the evaluation metrics with malware manifest features extracted using ECCD for Dataset 1. It can be seen that improved performance is obtained with models created using permissions, action attributes compared to all other features. An AUC of 0.985 and 0.720 was achieved using Random Forest classifier for (permissions and actions) attributes. The Random Forest classification algorithm out-

Table 8 Evaluation metrics for manifest features (**malware**) extracted using ***ECCD*** in **Dataset 1**

Training (573 malware + 879 benign)							
FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
H	36	1	0.504	1	0.5	1	0.504
P	556	75	**0.985**	60	0.96	70	0.877
A	2763	10	**0.720**	16	0.70	10	0.683
C	216	4	0.627	4	0.60	4	0.599
A_t	13243	78	0.716	20	0.65	40	0.642
S	1793	10	0.581	12	0.58	12	0.579
R	1994	19	0.607	10	0.59	19	0.6
P_r	342	1	0.502	1	0.5	1	0.502
Testing (381 malware + 585 benign)							
FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
P	556	75	**0.985**	60	0.96	70	0.877
A	2763	10	**0.720**	16	0.70	10	0.683

FS features, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve

Table 9 AUC for **benign** manifest features

Training (573 malware + 879 benign)							
FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
H	36	13	0.59	10	0.5	13	0.5
P	556	29	**0.81**	29	0.79	20	0.57
A	2763	40	**0.75**	40	0.73	47	0.70
C	216	6	0.65	6	0.5	6	0.5
A_t	13243	25	0.60	10	0.5	10	0.5
S	1793	2	0.50	2	0.5	2	0.5
R	1994	4	0.51	4	0.5	4	0.5
Testing (381 malware + 585 benign)							
FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
P	556	29	**0.81**	29	0.79	20	0.57
A	2763	40	**0.75**	40	0.73	47	0.70

FS features, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve

performed Rotation Forest and SVM. Similar trends were obtained during prediction phase when unlabelled vector were subjected to the learning models (refer Table 8).

Likewise, experiments were performed using features extracted from benign training set. Models constructed using permission and action attributes were reasonably better with respect to other attributes (refer Table 9).

5.2 Analysis with WI manifest features

The evaluation metrics for malware features is exhibited in Table 10. The performance of the models built using permissions and activities exceeds the models formed employing other attributes. Random Forest classifier resulted in an enhanced performance with AUC 0.994 (permissions) and 0.991 (activities). The prediction outcomes indicate that permissions and activities are the top two significant features (refer Table 10).

The experimental results for benign features is shown in Table 11. It can be observed from the table that the permissions and activity attributes produces acceptable results when compared to other features, but the performance of malware attributes is superior to benign.

5.3 Analysis with ECCD opcodes and strings

The performance with opcodes and strings extracted using ECCD for both malware and benign features is explained in the following sections.

Table 10 Evaluation metrics for manifest features (**malware**) extracted using ***WI*** in **Dataset 1**

Training (573 malware + 879 benign)							
FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
H	36	15	0.598	10	0.5	15	0.506
P	556	110	**0.994**	90	0.97	136	0.917
A	2763	250	0.914	90	0.85	250	0.787
C	216	22	0.747	10	0.60	22	0.605
A_t	13243	1450	**0.991**	90	0.82	1474	0.974
S	1793	274	0.873	40	0.74	274	0.868
R	1994	323	0.879	60	0.77	323	0.866
Pr	342	9	0.514	9	0.5	9	0.514
Testing (381 malware + 585 benign)							
FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
P	556	110	**0.994**	90	0.97	136	0.917
A_t	13243	1450	**0.991**	90	0.82	1474	0.974

FS features, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve

Table 11 Results for **benign** manifest features in terms of AUC on training and testing

Training (573 malware + 879 benign)							
FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
H	36	20	0.60	10	0.5	20	0.50
P	556	190	**0.99**	90	0.97	500	0.92
A	2763	2450	0.91	60	0.83	1850	0.77
C	216	170	0.77	10	0.60	140	0.60
A_t	13243	9500	**0.97**	40	0.79	11872	0.91
S	1793	1500	0.80	10	0.59	1000	0.58
R	1994	1694	0.83	40	0.64	1450	0.61
Pr	342	10	0.52	10	0.5	10	0.50
Testing (381 malware + 585 benign)							
FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
P	556	190	**0.99**	90	0.97	500	0.92
A_t	13243	9500	**0.97**	40	0.79	11872	0.91

FS features, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve]

The evaluation metrics for malware opcodes extracted using ECCD in prediction phase is depicted in Table 12. The results indicate that better performance is obtained with

Table 12 Evaluation metrics for opcodes in **prediction phase** using ***ECCD*** in **Datasets 1, 2** and **3**

Malware							
D	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
D1	211	20	**0.997**	20	0.994	20	0.57
D2	211	16	**0.994**	16	0.992	10	0.59
D3	211	9	**0.748**	9	0.733	9	0.55
Benign							
D	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
D1	211	100	**0.990**	60	0.984	100	0.833
D2	211	120	**0.987**	100	0.987	124	0.822
D3	211	80	**0.989**	100	0.987	124	0.825

D dataset, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve

Table 13 AUC for strings in **prediction phase** using ***ECCD*** in **Datasets 1, 2** and **3**

Malware							
D	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
D1	228	5	**0.959**	5	0.93	5	0.5
D2	225	8	**0.975**	8	0.96	8	0.52
D3	233	8	**0.974**	8	0.95	8	0.5
Benign							
D	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
D1	228	80	**0.993**	80	0.992	70	0.82
D2	225	76	**0.994**	76	0.993	76	0.81
D3	233	60	**0.988**	60	0.988	79	0.77

D dataset, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve

the Random Forest classification algorithm. The AUC is in range of 0.748 to 0.997. In the case of benign opcodes, it is found that Random Forest classifier resulted in increased performance compared to Rotation Forest and SVM (refer Table 12). The AUC for benign opcodes varies between 0.987 to 0.990.

The experimental results for the extraction of malware/benign strings through ECCD in prediction phase is demonstrated in Table 13. It is found from the table that improved performance is attained with Random Forest classifier with AUC in range of 0.959 to 0.975. The tabulated results for ECCD benign strings indicate that Random Forest classifier gives reasonable performance compared to others (refer Table 13). The AUC for ECCD benign strings varies between 0.988 to 0.994.

Table 14 Performance metrics for opcodes in **prediction phase** using ***WI*** in **Datasets 1, 2** and **3**

Malware							
D	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
D1	211	90	0.997	100	**0.997**	120	0.733
D2	211	30	0.996	80	**0.998**	40	0.732
D3	211	127	0.998	100	**0.998**	10	0.663
Benign							
D	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
D1	211	80	0.991	80	**0.991**	120	0.81
D2	211	100	0.986	127	**0.987**	127	0.82
D3	211	50	0.988	100	**0.988**	120	0.73

D dataset, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve

5.4 Analysis with WI opcodes and strings

The performance with opcodes and strings extracted using WI for malware/benign attributes is discussed in subsequent paragraphs.

The evaluation metrics for WI malware opcodes in prediction phase is presented in Table 14. It is observed from the table that highest AUC is achieved with Rotation Forest classifier, ranging between 0.997 to 0.998. The prediction results for benign opcodes (refer Table 14) indicate that an increased performance is observed with Rotation Forest classification algorithm. The AUC ranges between 0.987 to 0.991.

The performance metrics for malware strings extracted using WI in prediction phase is shown in Table 15. It can be noticed from the tabulated result that the Random Forest classifier outperformed Rotation Forest and SVM, with AUC ranging between 0.994 to 0.995. For WI benign strings, it was observed that Random Forest classifier resulted in reasonably accurate results (AUC in range of 0.988 to 0.995).

5.5 Heterogeneous feature space

The test results of the meta feature space model for both malware and benign features are discussed in this section.

The evaluation metrics for heterogeneous feature space developed using ECCD malware and benign features in the datasets are shown in Tables 16 and 17 respectively. It can be viewed from the table that relatively higher F-

Table 15 Evaluation metrics for strings in **prediction phase** using ***WI*** in **Datasets 1, 2** and **3**

Malware							
D	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
D1	228	130	**0.994**	120	0.993	148	0.900
D2	225	100	**0.995**	144	0.993	144	0.899
D3	233	155	**0.994**	80	0.994	150	0.905
Benign							
D	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
D1	228	150	**0.994**	80	0.994	165	0.900
D2	225	90	**0.995**	150	0.995	150	0.887
D3	233	90	**0.988**	110	0.994	165	0.898

D dataset, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve

measure is obtained for malware model compared its benign counterpart. Random Forest classifier resulted in enhanced performance in comparison to Rotation Forest and SVM. A F-measure of 0.972, 0.976 and 0.969 is achieved with malware features at feature lengths 198, 198 and 184 respectively for different datasets.

The performance metrics for heterogeneous feature space formed using malware and benign features for datasets are depicted in Tables 18 and 19 respectively. The tabulated results indicate that the malware model performs better. Based on the analysis of classifiers, it is observed that SVM (Linear Function) provides maximum F-measure as compared to others.

5.6 Result comparison

To substantiate the result effectiveness, the following comparisons were made (refer Table 20).

6 Discussions

Based on the investigation of experiments, we could infer the following:

1. What is the response of varying feature length on F-measure and AUC (Area Under ROC Curve)?
   From the conducted experiments it is observed that less attributes in the feature space result in poor performance. The primary reason is that these features do not represent the target classes. However, as the features are added in the feature space the variability in the feature vectors of instances belonging to the classes increases and thus the classification improves. Moreover, it is observed that beyond a certain limit the addition in feature length does not contribute to the formation of robust learning models. This is because irrelevant attributes appearing as noise increase misclassification rate.
2. Which classification algorithm result in improved performance?
   Classifiers such as Random Forest and SVM (Linear Function) resulted in better values of evaluation metrics

Table 16 Evaluation metrics for heterogeneous feature space developed using **malware** features (hardware, permission, action, category, activity, service, receiver and provider) obtained after applying ***ECCD*** in **Datasets 1, 2** and **3**

FL	Dataset	Random forest				Rotation forest				SVM (linear function)			
		TPR (%)	FPR (%)	F	T (s)	TPR (%)	FPR (%)	F	T (s)	TPR (%)	FPR (%)	F	T (s)
198	Dataset 1	**95.64**	**0.68**	**0.972**	**0.43**	89.00	2.39	0.924	6.07	85.17	5.46	0.880	0.20
198	Dataset 2	**96.14**	**0.57**	**0.976**	**0.45**	90.00	1.82	0.934	6.36	87.02	5.57	0.890	0.21
184	Dataset 3	**95.24**	**0.46**	**0.969**	**0.17**	84.76	0.57	0.910	4.33	83.81	2.05	0.884	0.15

FL feature length, *F* F-measure, *T* time in seconds

Table 17 Evaluation metrics with heterogeneous feature space formed using **benign** features (hardware, permission, action, category, activity, service, receiver and provider) obtained after applying ***ECCD*** in **Datasets 1, 2** and **3**

FL	Dataset	Random forest				Rotation forest				SVM (linear function)			
		TPR (%)	FPR (%)	F	T (s)	TPR (%)	FPR (%)	F	T (s)	TPR (%)	FPR (%)	F	T (s)
113	Dataset 1	94.76	18.43	0.850	0.29	85.86	21.84	0.783	3.94	82.02	28.67	0.726	0.24
119	Dataset 2	94.91	20.14	0.840	0.49	68.77	12.40	0.732	5.09	80.70	28.67	0.718	0.24
94	Dataset 3	75.87	10.01	0.745	0.21	35.24	1.59	0.504	2.18	20.32	2.39	0.320	0.16

Table 18 Performance metrics for heterogeneous feature space created using **malware** features (hardware, permission, action, category, activity, service, receiver and provider) obtained after applying ***WI*** in **Datasets 1, 2** and **3**

FL	Dataset	Random forest				Rotation forest				SVM (linear function)			
		TPR (%)	FPR (%)	F	T (s)	TPR (%)	FPR (%)	F	T (s)	TPR (%)	FPR (%)	F	T (s)
2437	Dataset 1	97.73	0.11	0.988	1.23	92.32	1.37	0.950	114.77	**99.30**	**0.46**	**0.993**	**0.70**
2379	Dataset 2	98.60	0.11	0.992	1.27	94.21	1.25	0.961	106.93	**99.47**	**0.46**	**0.994**	**0.70**
1886	Dataset 3	97.14	0.11	0.984	1.05	92.06	0.80	0.948	62.61	**99.05**	**0.23**	**0.992**	**0.54**

Table 19 Performance metrics for heterogeneous feature space designed using **benign** features (hardware, permission, action, category, activity, service, receiver and provider) obtained after applying ***WI*** in **Datasets 1, 2** and **3**

FL	Dataset	Random forest				Rotation forest				SVM (linear function)			
		TPR (%)	FPR (%)	F	T (s)	TPR (%)	FPR (%)	F	T (s)	TPR (%)	FPR (%)	F	T (s)
15398	Dataset 1	98.95	0.34	0.992	5.61	92.67	2.39	0.944	1594.82	97.21	0.80	0.980	1.79
15800	Dataset 2	99.65	0.34	0.996	5.09	92.28	2.62	0.940	1659.57	98.25	0.46	0.988	2.00
17628	Dataset 3	99.05	0.11	0.994	4.28	88.25	1.59	0.916	1500.46	98.41	0.11	0.990	1.78

Table 20 Comparison of Prior Works

Author	Remark
Kang et al. [23]	Detects malware with 98 % accuracy using features employing serial number of certificates. An accuracy of 90 % was achieved while performing classification of different malware families
Wang et al. [34]	Identified 40 dangerous permissions to flag samples as malicious with a detection accuracy of 94.62 % and a false positive rate of 0.6 %
Kang et al. [22]	Dalvik Bytecode frequency was analyzed for samples collected from Malware Genome project. Classification models were developed using Random Forest and an weighted average of 94 % F-measure was obtained
Canfora et al. [14]	The effectiveness of opcode *n-grams* was evaluated on apk files. *Bigrams* for 1000 opcodes resulted in 96.88 % accuracy with Random Forest
Our proposed method	Using ensemble attribute space an F-measure of 0.994 with TPR 99.47 % and FPR 0.46 % was attained for all datasets in 0.70 s

in less time with heterogeneous feature space (manifest features) using ECCD and WI approaches. Random Forest classifier resulted in higher AUC in both the feature selections. This is because Random Forest is an ensemble based classifier which decides the target class to be assigned to a given sample by aggregating the decision from multiple forests. This implies the class is assigned to a specimen based on aggregate vote.

3. Which learning model effectively identifies the unseen samples?
   Models generated using malware features contribute largely towards classifying the samples due to the diverse nature of the benign applications. Thus, consolidating a common behaviour from benign specimens is very difficult. However, malware files are also diverse but they are not as complex as their legitimate counterparts. This is because few instructions in the form of opcodes, permissions and application components are to be retained in order to preserve degree of malice.
4. Which feature selection technique provides the best results?
   While comparing the feature selection techniques, it is found that better performance is obtained for Weighted Mutual Information (WI). This is concluded by investigating the feature vectors. In comparison to ECCD the samples depicted higher variance in the feature vectors and thus showed higher performance.

7 Conclusions

A static analysis framework for identifying malicious Android applications is proposed by extracting features from the manifest and dex files. Initial pruning is performed to eliminate irrelevant attributes from higher dimension feature space. Then feature selection methods such as ECCD and WI were applied to determine significant features for generating learning models. Training models are generated with individual categories of features and the performance is evaluated using F-measure and AUC (Area Under ROC Curve). Prominent attributes were subsequently aggregated to form the ensemble feature space (constituting manifest file components). The performance evaluation with ensemble features showed an improvement over individual attribute space. An

overall analysis demonstrated that the malware model outperformed the benign model. An F-measure of 0.972 (Dataset 1), 0.976 (Dataset 2) and 0.969 (Dataset 3) was obtained with meta feature space model with malware features using ECCD for Random Forest classifier. For WI, an F-measure of 0.993 (Dataset 1), 0.994 (Dataset 2) and 0.992 (Dataset 3) was achieved with SVM (Linear Function) classifier. Comparison of feature selection methods depicted superiority of WI over ECCD. Future work is expected to investigate malwares showing different behavioural patterns on native and virtual machines. Also a malware normalizer for handling obfuscation would be considered for achieving better results that can identify a zero day vulnerability.

Appendix: Test results for dataset 2 and dataset 3

See Tables 21, 22, 23, 24, 25, 26, 27 and 28.

Table 21 Performance with manifest features (**malware**) in **prediction phase** using ***ECCD*** in **Dataset 2**

FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
P	551	68	**0.987**	68	0.965	68	0.865
A	2725	17	**0.752**	17	0.733	17	0.709

FS features, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve

Table 22 Evaluation metrics for manifest features (**malware**) in predicting samples in **test set** using ***ECCD*** in **Dataset 3**

FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
P	561	60	**0.990**	50	0.964	65	0.869
A	2707	19	**0.742**	10	0.708	19	0.563

FS features, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve

Table 23 Evaluation metrics with manifest features (**benign**) in **predicting samples** using ***ECCD*** in **Dataset 2**

FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
P	551	20	**0.823**	20	0.794	30	0.510
A	2725	43	**0.754**	20	0.721	43	0.683

FS features, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve

Table 24 Evaluation metrics for manifest features (**benign**) in **prediction phase** using ***ECCD*** in **Dataset 3**

FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
P	561	29	**0.820**	29	0.755	10	0.5
A	2707	37	**0.761**	10	0.700	37	0.506

FS features, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve

Table 25 Performance with manifest features (**malware**) in **prediction phase** using ***WI*** in **Dataset 2**

FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
P	551	100	**0.997**	90	0.983	126	0.917
A_t	13190	1400	**0.996**	90	0.826	1435	0.981

FS features, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve

Table 26 Evaluation metrics for manifest features (**malware**) in predicting samples in **test set** using ***WI*** in **Dataset 3**

FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
P	561	130	**0.996**	50	0.986	100	0.935
A_t	13002	1100	**0.997**	100	0.846	1100	0.983

FS features, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve

Table 27 Evaluation metrics with manifest features (**benign**) in predicting samples in **test set** using ***WI*** in **Dataset 2**

FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
P	551	120	**0.99**	60	0.98	350	0.924
A_t	13190	10000	**0.97**	30	0.80	11872	0.920

FS features, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve

Table 28 Evaluation metrics for manifest features (**benign**) in **prediction phase** using ***WI*** in **Dataset 3**

FS	N	Ran. F.		Rot. F.		SVM (LF)	
		FL	AUC	FL	AUC	FL	AUC
P	561	190	**0.99**	60	0.98	510	0.92
A_t	13002	11872	**0.98**	40	0.81	11872	0.95

FS features, *N* total features, *FL* pruned feature length, *AUC* area under ROC curve

References

1. Androguard. http://code.google.com/p/androguard/. Accessed 7 Dec 2014
2. Android Malware Genome Project. http://www.malware-genomeproject. Accessed 5 May 2014
3. Apk File Format. http://www.file-extensions.org/article/android-apk-file-format-description. Accessed 7 Dec 2014
4. Drebin Dataset. http://user.cs.uni-goettingen.de/~darp/drebin/. Accessed 2 Jan 2015
5. Gartner. http://www.gartner.com/newsroom/id/3010017. Accessed 8 Jan 2015
6. Google Play store. https://play.google.com/store?hl=en. Accessed 5 May 2014
7. Smali/Baksmali. http://code.google.com/p/smali/. Accessed 5 May 2014
8. WEKA-Open Source Machine Learning Software. http://www.cs.waikato.ac.nz/ml/weka. Accessed 8 Jan 2015
9. Aafer, Y., Du, W., Yin, H.: DroidAPIMiner: mining API-level features for robust malware detection in Android. In: Security and Privacy in Communication Networks, pp. 86–103. Springer, Berlin (2013)
10. Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., Rieck, K.: Drebin: effective and explainable detection of Android malware in your pocket. In: Proceedings of the Annual Symposium on Network and Distributed System Security (NDSS) (2014)
11. Aung, Z., Zaw, W.: Permission-based Android malware detection. Int. J. Sci. Technol. Res. **2**(3), 228–234 (2013)
12. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
13. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: behavior-based malware detection system for Android. In: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, pp. 15–26, ACM (2011)
14. Canfora, G., De Lorenzo, A., Medvet, E., Mercaldo, F., Visaggio, C.A.: Effectiveness of opcode ngrams for detection of multi family Android malware. In: 2015 10th International Conference on Availability, Reliability and Security (ARES), pp. 333–340. IEEE (2015)
15. Chakradeo, S., Reaves, B., Traynor, P., Enck, W.: MAST: Triage for market-scale mobile malware analysis. In: Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks, pp. 13–24. ACM (2013)
16. Chang, C.C., Lin, C. J.: LIBSVM: a library for support vector machines. ACM Trans. Intell. Syst. Technol. **2**(3), 1–27 (2011)
17. Enck, W., Ongtang, M., McDaniel, P.D.: On lightweight mobile phone application certification. In: ACM Conference on Computer and Communications Security, pp. 235–245. ACM (2009)
18. Feizollah, A., Anuar, N.B., Salleh, R., Wahab, A.: A review on feature selection in mobile malware detection. Digital Investig. **13**, 22–37 (2015)
19. Felt, A.P., Chin, E., Hanna, S., Song, D., Wagner, D.: Android permissions demystified. In: Proceedings of ACM Conference on Computer and Communications Security (CCS), pp. 627–638 (2011)
20. Felt, A.P., Finifter, M., Chin, E., Hanna, S., Wagner, D.: A survey of mobile malware in the wild. In: Proceedings of ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM), pp. 3–14 (2011)
21. Feng, Y., Anand, S., Dillig, I., Aiken, A.: Apposcopy: semantics-based detection of Android malware through static analysis. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 576–587 (2014)
22. Kang, B., Kang, B.J., Kim, J., Im, E.G.: Android malware classification method: Dalvik bytecode frequency analysis. In: Proceedings of the 2013 research in adaptive and convergent systems, pp. 349–350. ACM (2013)
23. Kang, H., Jang, J.-W., Mohaisen, A., Kim, H.K.: Detecting and classifying Android malware using static analysis along with creator information. Int. J. Distrib. Sens. Netw. (2015). doi:10.1155/2015/479174
24. Kim, D., Kwak, J., Ryou, J.: DWroidDump: executable code extraction from Android applications for malware analysis. Int. J. Distrib. Sens. Netw. (2014). doi:10.1155/2015/379682
25. Largeron, C., Moulin, C., Géry, M.: Entropy based feature selection for text categorization. In: Proceedings of the 2011 ACM Symposium on Applied Computing, pp. 924–928. ACM (2011)
26. Peng, H., Gates, C.S., Sarma, B.P., Li, N., Qi, Y., Potharaju, R., Nita-Rotaru, C., Molloy, I. : Using probabilistic generative models for ranking risks of Android apps. In: ACM Conference on Computer and Communications Security, pp. 241–252. ACM (2012)
27. Rastogi, V., Chen, Y., Jiang, X.: DroidChameleon: evaluating Android anti-malware against transformation attacks. In: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, pp. 329–334. ACM (2013)
28. Rodriguez, J.J., Kuncheva, L.I., Alonso, C.J.: Rotation forest: a new classifier ensemble method. IEEE Trans. Pattern Anal. Mach. Intell. **28**(10), 1619–1630 (2006)
29. Sarma, B.P., Li, N., Gates, C.S., Potharaju, R., Nita-Rotaru, C., Molloy, I.: Android permissions: a perspective combining risks and benefits. In: Proceedings of the 17th ACM Symposium on Access Control Models and Technologies, pp. 13–22. ACM (2012)
30. Schaffernicht, E., Gross, H.-M.: Weighted mutual information for feature selection. In: Artificial Neural Networks and Machine Learning-ICANN, pp. 181–188. Springer, Berlin (2011)
31. Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y.: Andromaly: a behavioral malware detection framework for Android devices. J. Intell. Inf. Syst. **38**(1), 161–190 (2012)
32. Spreitzenbarth, M., Freiling, F. C., Echtler, F., Schreck, T., Hoffmann, J.: Mobile-sandbox: having a deeper look into Android applications. In: Proceedings of the 28th Annual ACM Symposium on Applied Computing, pp. 1808–1815. ACM (2013)
33. Varsha, M.V., Vinod, P., Dhanya, K.A.: Heterogeneous feature space for Android malware detection. In: Proceedings of 8th IEEE International Conference on Contemporary Computing (IC3-2015), Jaypee Institute of Information Technology, Noida (2015)
34. Wang, W., Wang, X., Feng, D., Liu, J., Han, Z., Zhang, X.: Exploring permission-induced risk in Android applications for malicious application detection. IEEE Trans. Inf. Forensics Secur. **9**(11), 1869–1882 (2014)
35. Wu, D.-J., Mao, C. -H., Wei, T. -E., Lee, H. -M., Wu, K. -P.: DroidMat: Android malware detection through manifest and API calls tracing. In: Proceedings of Asia Joint Conference on Information Security (Asia JCIS), pp. 62–69 (2012)
36. Zhou, Y., Jiang, X.: Dissecting Android malware: characterization and evolution. In: IEEE Symposium on Security and Privacy, pp. 95–109. IEEE Computer Society (2012)
37. Zhang, M., Duan, Y., Yin, H., Zhao, Z.: Semantics-aware Android malware classification using weighted contextual API dependency graphs. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS '14), pp. 1105–1116 (2014)
38. Chinese market. http://shouji.baidu.com/s?wd=shareit&data_type=app&f=header_app. Accessed 5 May 2014
39. Android App. Store. http://www.9apps.com/. Accessed 5 May 2014
40. http://contagiodump.blogspot.in/. Accessed 15 Jan 2015
41. Alternate App stores. http://www.ubergizmo.com/articles/google-play-store-alternatives/. Accessed 5 May 2014