




# A multiview learning method for malware threat hunting: windows, IoT and android as case studies

Hamid Darabian<sup>1</sup> · Ali Dehghantanha<sup>2</sup> · Sattar Hashemi<sup>1</sup> · Mohammad Taheri<sup>1</sup> · Amin Azmoodeh<sup>2</sup> · Sajad Homayoun<sup>3</sup> · Kim-Kwang Raymond Choo<sup>4</sup>  · Reza M. Parizi<sup>5</sup>

Received: 17 December 2018 / Revised: 10 October 2019 / Accepted: 28 October 2019 /  
Published online: 17 January 2020  
© Springer Science+Business Media, LLC, part of Springer Nature 2020

## Abstract

Malware remains a threat to our cyberspace and increasingly digitalized society. Current malware hunting techniques employ a variety of features, such as OpCodes, ByteCodes, and API calls, to distinguish malware from goodware. However, existing malware hunting approaches generally focus on a single particular view, such as using dynamic information or opcodes only. While single-view malware hunting systems may provide lean and optimized basis for detecting a specific type of malware, their performance can be significantly limited when dealing with other types of malware; thus, making it trivial for an advanced attacker to develop malware that simply obfuscates features monitored by a single-view malware detection system. To address these limitations, we propose a multi-view learning method that uses multiple views including OpCodes, ByteCodes, header information, permission, attacker's intent and API call to hunt malicious programs. Our system automatically assigns weights to different views to optimize detection in different environment. Using experiments conducted on various Windows, Android and Internet of Things (IoT) platforms, we demonstrate that our method offers high accuracy with a low false positive rate on these case study platforms. Moreover, we also investigate the robustness of detection against weak views (features with low power of discrimination). The proposed method is the first malware threat hunting method that can be applied to different platforms, at the time of this research, and it is considerably difficult for attackers to evade detection (since it requires attackers to obfuscate multiple different views).

**Keywords** Malware · Threat hunting · Malware detection · Multi-view learning · Maximum margin · View weighting

---

This article belongs to the Topical Collection: *Special Issue on Smart Computing and Cyber Technology for Cyberization*

Guest Editors: Xiaokang Zhou, Flavia C. Delicato, Kevin Wang, and Runhe Huang

---

✉ Sattar Hashemi  
s\_hashemi@shirazu.ac.ir

Extended author information available on the last page of the article.

## 1 Introduction

As cyber attacks become more frequent and common place, the research community has presented a number of different approaches (e.g. heuristic methods) to detect and analyze malicious programs (also referred to as malware) [20, 35, 39]. However, designing effective and efficient malware detection approaches remains challenging [3, 24, 48], for example due to counter-malware detection efforts by malware authors and cyber criminals. Examples of approaches to circumvent the detection of malware detection systems include the use of obfuscation techniques, where malware samples are modified and mutated in order to deceive host-based and network-based detection systems [16, 40, 47, 50]. Specifically, these obfuscation techniques alter the structure of malicious codes or run-time behavior of the malware despite keeping the malicious functionality. This can be achieved using techniques such as code encryption or changing the sequence and order of library calls [47, 49].

There are three major approaches for analyzing malware namely: static, dynamic and hybrid [8, 9]. Static approaches generally rely on static feature extraction from executable, such as API calls, OpCodes, ByteCodes, and header information, for malware analysis. On the other hand, dynamic analysis techniques are based on the execution of suspicious samples to generate behavioral features from system calls, captured network traffics, resource consumption pattern, etc. Although static analysis approaches are more straightforward, it is (significantly) less effective against code obfuscations. On the other hand, dynamic analysis techniques are relatively resource consuming and demand more time, but they are generally more effective [31]. Malware authors are also known to continually designed circumvention techniques, such as anti-disassembly, anti-virtual machine (VM), and anti-debugging, to evade malware detection [25, 38, 44]. Therefore, there have been attempts to design hybrid malware analysis approaches, in order to leverage features obtained from both static and dynamic analysis to enhance the robustness of detection methods [34]. However, these techniques are generally in their infancy and most merging of static and dynamic feature activities are manual and time/resource consuming.

Multi-view learning approaches are a promising solution to mitigate some of the limitations that underpin static, dynamic and hybrid malware detection approaches [12, 22, 45, 52]. Multi-view nature data can be acquired in real world applications, to form disparate feature sets. Specifically, each of these feature sets is referred to as a view, which can be obtained from multiple sources or different feature subsets to demonstrate a unique semantic perspective of data. A particular single view may not adequately express or present the information of all samples comprehensively. One naive solution for model construction on multiple views of a data is to concatenate all of the multiple views to generate a single view for learning algorithms [51]. However, limitations of such a naive approach include overfitting on small training sets, the need for feature engineering, and dependency of the model on the underlying data. Efficient multi-view learning methods can generally learn a function on each view, and optimize or integrate functions in order to enhance performance of the model [42].

To adapt multi-view learning for malware threat hunting, we posit the potential of using executable files for both malware or goodware (i.e. non-malware) as the source for multi-view data extraction. The challenge, however, is to determine what algorithm can be used to effectively extract features from executable files. These feature sets are generally extracted from file header, ByteCodes, API calls, OpCodes, system calls and captured network traffic during run time [23]. As discussed earlier, existing approaches generally use single view information for malware detection or a combination of some views in the form of a single view [26, 34, 37].

In this paper, we propose an efficient ensemble multi-view learning method, which automatically assigns optimized weights to different views. The proposed method is a large margin classifier that improves generalization using a global optimum hypothesis [17]. Additionally, we introduce a new constraint programming method to perform view weighting as these weights play a crucial role in our proposed ensemble. Since our optimization problem is convex, we can always find a global optimum point.

In order to evaluate performance of our proposed approach, we use the typical machine learning metrics, namely: True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). TP demonstrates malware samples that are correctly predicted, TN denotes the number of samples that are correctly identified as benign, FP reflects goodware samples that are incorrectly detected as malware, and FN shows malware samples that are incorrectly predicted as a goodware. Accuracy, precision, recall and f-measure metric are then calculated using these metrics. Specifically, *accuracy* (see (1)) is the proportion of correct outcomes (malware and goodware) out of the observed samples, *precision* of a classifier reflects the ratio of correctly predicted malware samples to the total predicted malware observation (see (2)), *recall* or sensitivity is the fraction of successful prediction of relevant malware samples (see (3)), and *F-measure* indicates the overall performance of a classification algorithm based on a combination of *Precision* and *Recall* (see (4)).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

$$F - measure = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (4)$$

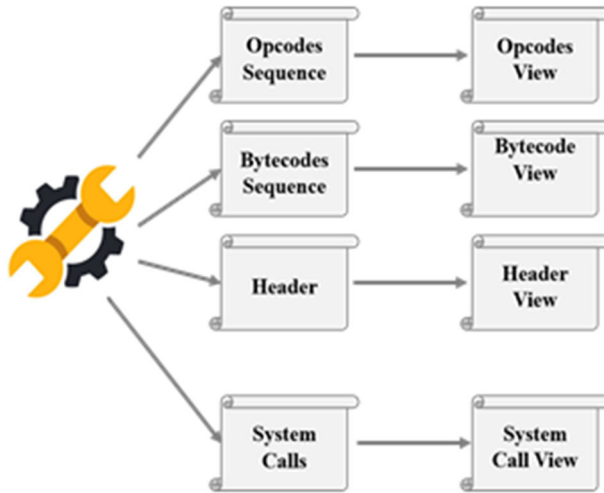
Furthermore, we will report Receiver Operating Characteristics (*ROC*), Area Under the Curve (*AUC*), and Matthews Correlation Coefficient (*MCC*) of our classifiers as well. *ROC* compares different classifiers independently regardless of their class distribution or cost. *ROC* curve plots *TP* rate against *FP* rate for different thresholds. *AUC* is a metric designed for evaluating classifier's output quality. *AUC* is calculated based on the area of the convex shape below the *ROC* curve between 0 and 1, where 1 shows the perfect prediction capability and *MCC* provides a measurement of the quality for binary classifications to mitigate imbalanced dataset side effect(see (5)).

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(FP + TP)(FN + TP)(FP + TN)(TN + FN)}} \quad (5)$$

The rest of this paper is structured as follows: Section 2 describes the datasets, feature extraction and view generation method, used in our proposed approach and its evaluation. Section 3 presents the proposed method, and Section 4 discusses the empirical evaluation findings. Specifically, the findings suggest that the proposed classification method improves the generalization performance of the learning model, with an acceptable accuracy rate. Sections 5 and 6 respectively present the related literature and conclusion.

## 2 Datasets and view generation

A variety of features can be used to efficiently detect malicious programs, ranging from file header information to OpCodes to ByteCodes to API calls to system call sequences

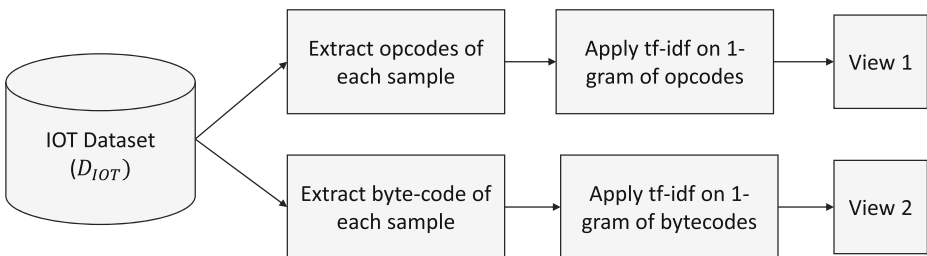


**Figure 1** Extracting various views from a program (malicious or normal)

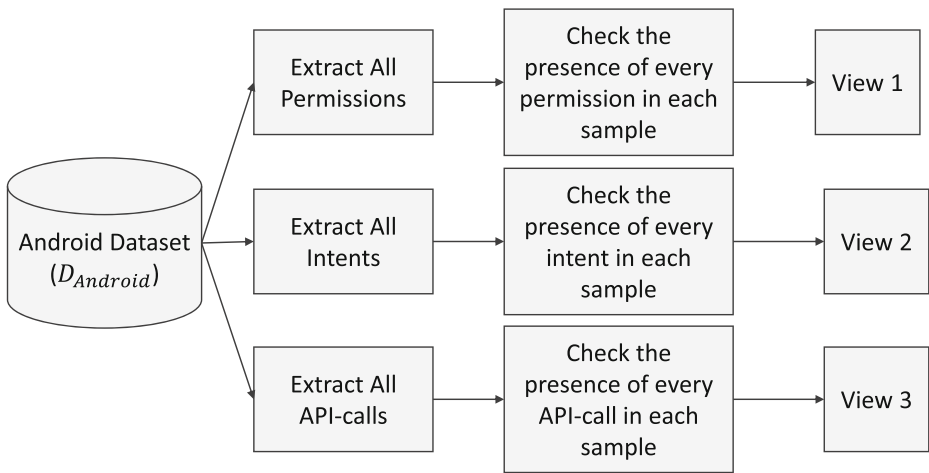
to access privilege, and so on. We can build different views using each of these features to test our multi-view learning algorithm (see Figure 1). Before applying a multi view-learning algorithm, each view should be converted into machine-understandable features [35]. Moreover, to demonstrate adaptability of our proposed approach in detecting malware on different platforms, we evaluate our system with different datasets comprising Internet of Things (IoT), Windows and Android malware. In this section, we will describe these datasets and the obtained views for each dataset.

**IoT dataset** The IoT malware and benign application dataset, as described in our earlier work [2, 14], includes malware and goodware binary files of various IoT platforms. Specifically, the dataset includes 280 malware and 271 goodware files compiled for ARM processor. Two views are generated for malware in this dataset, namely: OpCodes and ByteCodes views. Figure 2 described the procedure used to construct these two views.

To extract opcodes, each sample was unpacked using Debian installer bundle and Object-Dump [14]. Then, by considering the OpCodes of each sample as a sequence and utilizing the information retrieval methods such as term frequency–inverse document frequency (tf-idf), we converted textual OpCode dataset to a numerical format. Tf-idf is a popular



**Figure 2** Views of IoT dataset



**Figure 3** Views of the Drebin dataset

numerical technique to highlight the importance of a word (in our context, OpCode) in a textual dataset [5]. In this research, we applied tf-idf in the 1-gram setting [35].

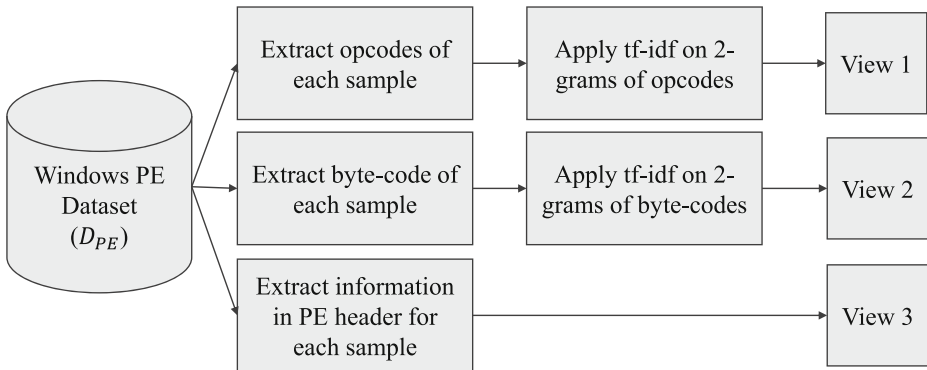
The minimal part of an executable sample is ByteCode. Similar to the OpCode sequence format, we generated N-grams tf-idf for ByteCode, which is a prevalent approach for machine learning-based static malware analysis [29]. There are various tools to extract ByteCodes, such as hexdump. Once ByteCode sequence of samples are extracted, tf-idf method can be used to convert the sequence into 1-gram ByteCode numerical vectors [35].

**Android Dataset** In order to evaluate the utility of our proposed approach in detecting Android malware, the Drebin benchmark dataset<sup>1</sup>[1] was used. This dataset contains 123,453 goodware applications from different markets and 5,560 malware samples from 179 family of malicious code. The extracted features are organized in sets of permissions, API calls, network addresses, intents, content providers, etc. (see Figure 3).

Application privilege is essential part of the Android security mechanism. Malicious applications tend to request access to specific permissions more often than a benign application [33]. Using privilege information available on the Drebin dataset, an indicator vector is generated for each sample that a ‘1’ value represents the application requesting for some permission and a ‘0’ value otherwise. The API call feature denotes the request to access and execute specific API of the Android operating system. Since misusing these APIs is a prevalent technique in Android malware [32], we collected sensitive API calls and considered them as a particular view for this dataset, and the feature vector was generated in a manner similar to the Permissions data. Intention information in Android is designed to establish communication and for exchanging passive data between processes. We collected all intents in the Drebin dataset to generate another view.

**PE windows dataset** We also used the 2860 executable and Dynamic Link Library(DLL) of Microsoft Windows samples, which include 1329 malware and 1531 benign. Malware samples are a mixture of packed, not packed and metamorphic files, which are randomly

<sup>1</sup>publicly available at <https://www.sec.cs.tu-bs.de/~danarp/drebin/download.html>



**Figure 4** Views of PE files

selected from the *VXHeaven* dataset. To prepare benign samples, a collection of known good executable and DLL were collected from a standard Windows installation and their hashes were cross-checked. ByteCodes, OpCodes, and PE header are three extracted views for Windows files are presented in Figure 4, and we remark that the PE header contains valuable information [27]. We considered the structure defined in the Windows header files as features, which can be extracted with different tools such as *pefile* library<sup>2</sup>.

The views of datasets are publicly available on <https://cybersciencelab.org/multiview-dataset/>.

### 3 Proposed approach

For a multi-view learning problem, there is  $|v|$  views (or feature sets) for each dataset. The collection of views is denoted as  $X = \langle X^1, X^2, \dots, X^v, y \rangle$ , where  $X^i$  is  $i^{th}$  view and  $y$  is a vector to demonstrate the samples' labels.  $X_p^i$  is the feature vector of the  $p^{th}$  training pattern in the  $i^{th}$  view, and  $y_p$  is its corresponding class label that can be +1 and -1 for malware and benign samples respectively. Each of these views is leveraged to distinguish between the samples. Therefore, it is crucial to indicate more significant views by assigning weights to them. The Support Vector Machine(SVM) [15] partly inspires our method for assigning weights. Finally, we will have a multi-view ensemble classifier that trains our proposed model according to calculated weights. The proposed model includes two major phases, namely: the training and Testing Phase. Figure 5 depicts an overall information about the method.

Ensemble models tend to improve the generality of classification approaches using a combination of classifiers. So, the proposed model trains a classifier for each view and considers every trained model as a membership function or  $MF_i$  (Figure 6), where the  $MF_i$  function specifies how much a pattern  $X_p^i$  is compatible with a class. The outputs of  $MF_i$  for sample  $X_p^i$  is  $\mu_+^i(X_p^i)$  for a membership value of the positive class and  $\mu_-^i(X_p^i)$  for a membership value of the negative class. We addressed the  $\mu_+^i(X_p^i)$  and  $\mu_-^i(X_p^i)$  as the compatibility grades of pattern  $X_p^i$  with model  $i^{th}$ .

<sup>2</sup><https://github.com/erocarrera/pefile>

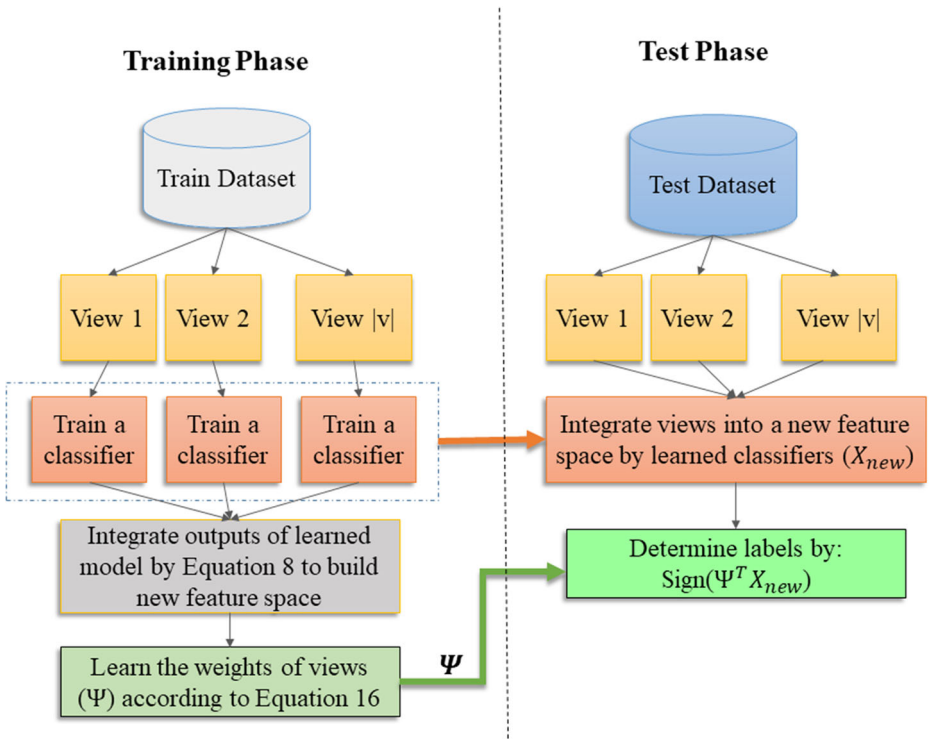


Figure 5 The proposed model diagram

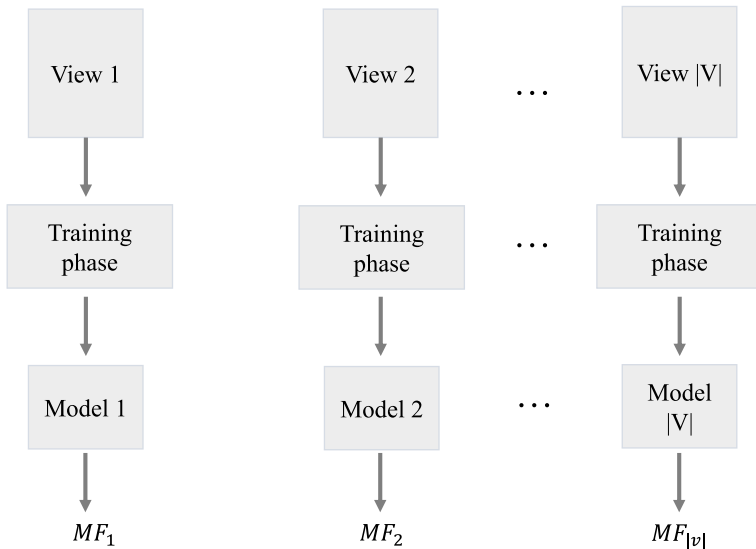


Figure 6 Membership functions for each view

Every trained model calculates  $\mu_{+}^i(X_p^i)$  and  $\mu_{-}^i(X_p^i)$  for each sample  $X_p^i$ . To have a unit classifier, we integrated the result of models based on  $\mu_{+}^i(X_p^i)$  and  $\mu_{-}^i(X_p^i)$  for all training samples. Moreover, we deemed the importance of trained model on each view by assigning specific weights to them. Figure 7 briefly illustrates the proposed weighting method.

According to Figure 7, the weights of views are not initially determined. Therefore, we should determine these such that the maximum training patterns will be correctly classified.  $C_{(i)}$  is the  $i^{th}$  classifier which is trained on the  $i^{th}$  view. Each sample  $X_p$  is classified correctly by means of weighted voting, if (6) is satisfied.

$$y_p(\sum_{C_i} (W_{i,+} \times \mu_{+}^i(X_p^i)) - \sum_{C_i} (W_{i,-} \times \mu_{-}^i(X_p^i))) > 0$$

$$y_p = +1 \quad \text{or} \quad y_p = -1 \tag{6}$$

Tuning the weights to satisfy (6) is not possible for all training patterns. To overcome this problem, we propose a method to increase the size of margin of separation in order to improve the generalization inspired by SVM for the view weighing process. The decision boundary which is applied by SVM[15, 41] separates the space between +1 and -1 classes. All points on decision boundary have an equal value of  $f(X)$ . The proposed models defines the decision boundary  $H$  as a sub-space such that all its points have the same compatibility grade or in another words, they have equal strength. Here, the distance of a point to the decision boundary is considered as the strength of correct class minus the strength of the opposite class. Hence, the discriminant function  $\chi(X)$  is defined as shown in (7).

$$\chi(X) = \text{sign}(\sum_{C_i} (W_{i,+} \times \mu_{+}^i(X_p^i)) - \sum_{C_i} (W_{i,-} \times \mu_{-}^i(X_p^i))) = \pm 1 \tag{7}$$

For simplicity and a better understanding of our proposed method, we define a vector demonstrating the weights of views as  $\psi = [w_{1,+}, w_{2,+}, \dots, w_{v,+}, w_{1,-}, \dots, w_{v,-}]$  and

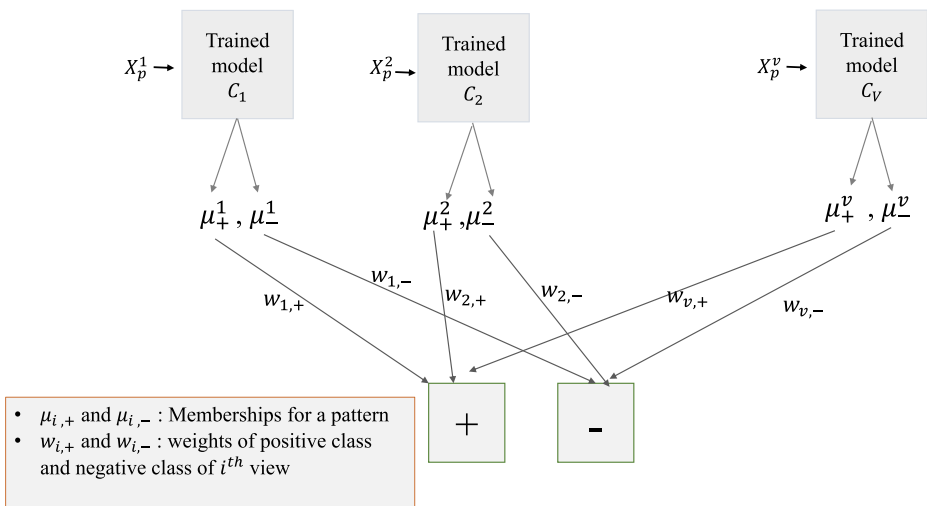


Figure 7 Overview of proposed model



transfer each pattern  $X_p = \langle X_{p1}, X_{p2}, \dots, X_{pv} \rangle$  by  $M(X)^T$  to a new feature space such that the new representation includes a signed compatibility grade, which is defined in (8).

$$M(X_p)^T = [\mu_+^1(X_p^1), \mu_+^2(X_p^2), \dots, \mu_+^v(X_p^v), -1 \times \mu_-^1(X_p^1), -1 \times \mu_-^2(X_p^2), \dots, -1 \times \mu_-^v(X_p^v)] \tag{8}$$

Therefore, the decision boundary  $H$ , in (7) can be written as (9) based on the transformation which is a discriminant hyper-plane crossing the origin.

$$H : \chi(X) = \psi^T M(X) = 0 \tag{9}$$

To calculate  $\psi$ , we take similar approach to the SVM’s perpendicular vector for discriminant hyperplane. Hence,  $\psi$  can be obtained by considering it as the perpendicular vector of the discriminant hyperplane of the compatibility space. At the beginning, training patterns are transferred to a compatibility grade space by using the transfer function  $M(X)$ . Afterwards, a discriminant hyperplane as the decision boundary is calculated. Although any linear classifier can find this hyperplane, we opted the SVM based on its high generalization capability. Therefore, the perpendicular vector of the found decision boundary is the desired vector  $\psi$ , which is the weights of its corresponding views. Accordingly, the view weighting problem can be presented as shown in (10).

$$\begin{aligned} & \text{Minimize } \frac{1}{2} \Psi^2 + C \sum_p \epsilon_p \\ & \text{s.t. } \forall X_p : \Psi^T M(X_p) y_p \geq 1 - \epsilon_p \\ & \epsilon_p \geq 0 \end{aligned} \tag{10}$$

Unlike SVM, this optimization problem discriminant hyperplane is forced to contain the origin. It should be emphasized that the margin can be maximized as per (10).

The proposed method for view weighting is similar to an unbiased SVM method with a new kernel function. A transfer function of unbiased SVM is  $\phi(X)$  [6] while the transfer function in the proposed method is  $M(X)$ . Hence, the proposed kernel for our ensemble multi-view classifier can be defined as shown in (11). The kernel has a considerable difference over previous kernels (RBF, Polynomial, Sigmoid, etc.). While the previous kernels are static and calculated based on the dot product of two patterns, the proposed method’s kernel is dynamic because it is derived from trained classifiers using multi-view data. Here,  $M(X_p)$  is transferred sample  $X_p$  to the new multiview space using (8) and  $K(X_p, X_q)$  is kernel on the new multiview space that we optimize it to achieve best performance for the proposed method.

$$K(X_p, X_q)^* = M(X_p)^T M(X_q) \tag{11}$$

It is worth noting that some elements of the vector of view weights ( $\psi$ ) may be negated based on the value given to parameter  $C$  because it is a regularization parameter. If we define a condition that weights can not be negative, we can add a constraint ( $\forall w_i \text{ in } \psi : w_i \geq 0$ ) to the optimization problem in (10) to overcome the problem of negative weights [43]. Hence, the optimization problem would be similar to (12).

$$\begin{aligned} & \text{Minimize } \frac{1}{2} \Psi^2 + C \sum_p \epsilon_p \\ & \text{s.t. } \forall X_p : \Psi^T M(X_p) y_p \geq 1 - \epsilon_p \\ & \epsilon_p \geq 0 \\ & \forall w_i \in \psi : w_i \geq 0 \end{aligned} \tag{12}$$

Algorithm 1 and Algorithm 2 describe the proposed method’s pseudo-code for training and test phase respectively.

**Algorithm 1** Training phase.

---

**Data:** Dataset\_train  
**Result:** Calculate  $\Psi$  vector as weights of views  
 classifiers = a set of base classifiers  
 $V$  = Number of views  
 $N$  = Number of samples in Dataset\_train  
 trained\_classifiers = {}  
 $X = [V][N][:]$   
 $M = [N][2 \times V]$  //new feature space  
**for**  $p$  in Dataset\_train **do**  
   **for**  $v$  in range(1, $V$ ) **do**  
      $X[v][p][:] = \text{extract\_features}(\text{sample } p^{th}, v)$  //extracts features from sample  $p^{th}$  in view  $v^{th}$   
   **end for**  
**end for**  
**for**  $v$  in range(1, $V$ ) **do**  
    $c_i = \text{train\_classifier}(X[i][:], \text{classifiers}(j))$  // trains classifier  $j^{th}$  on view  $i^{th}$   
   trained\_classifiers.append( $c_i$ )  
**end for**  
**for**  $v$  in range(1, $V$ ) **do**  
   **for** training\_pattern( $p^{th}$ ) in  $X[v][:][:]$  **do**  
      $M[p][v] = \text{get\_positive\_membership}(\text{trained\_classifiers}(v), X[v][p][:])$   
      $M[p][2 \times v] = \text{get\_negative\_membership}(\text{trained\_classifiers}(v), X[v][p][:])$   
   **end for**  
**end for**  
 calculate  $\Psi$  vector according to (12)

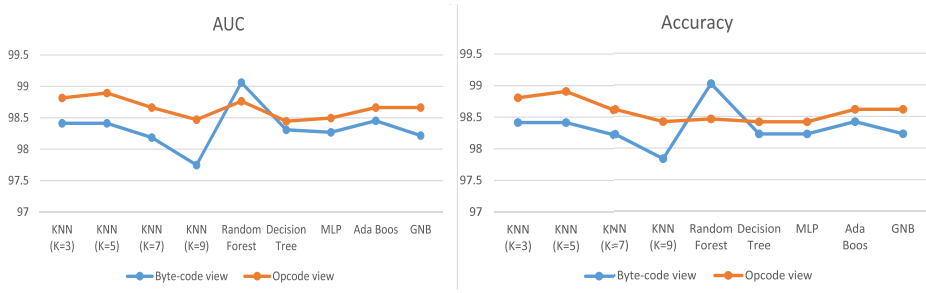
---

**Algorithm 2** Test phase.

---

**Data:** Dataset\_test,  $\Psi$ , trained\_classifiers,  $V$   
**Result:** Labels of samples in Data\_test  
 $N$  = Number of samples in Dataset\_test  
 $X_{\text{test}} = [V][N][:]$   
 $M_{\text{test}} = [N][2 \times V]$  //new feature space  
**for**  $p$  in Dataset\_test **do**  
   **for**  $v$  in range(1, $V$ ) **do**  
      $X_{\text{test}}[v][p][:] = \text{extract\_features}(\text{sample } p^{th}, v)$  //extract features from sample  $p^{th}$  in view  $v^{th}$   
   **end for**  
**end for**  
**for**  $v$  in range(1, $V$ ) **do**  
   **for** test\_pattern( $p^{th}$ ) in  $X_{\text{test}}[v][:][:]$  **do**  
      $M_{\text{test}}[p][v] = \text{get\_positive\_membership}(\text{trained\_classifiers}(v), X_{\text{test}}[v][p][:])$   
      $M_{\text{test}}[p][2 \times v] = \text{get\_negative\_membership}(\text{trained\_classifiers}(v), X_{\text{test}}[v][p][:])$   
   **end for**  
**end for**  
 calculate label vector by  $\text{sign}(\Psi^T M_{\text{test}})$

---



**Figure 8** Accuracy and AUC of trained classifiers on each view of IoT dataset

### 4 Results and discussion

We should first train a classifier for every view of each dataset and then integrate the results of trained classifiers in the manner described in the previous section. Although we can use any desired classifier on each view, we decided to do an exploratory study to find the most suitable classifier for our tasks. Furthermore, we have used 10-fold Cross-Validation[19] to evaluate our method. Figure 8 displays the *Accuracy* and *AUC* of classifiers on OpCode and ByteCode views of  $D_{IoT}$  dataset. Therefore, *KNN* (K=5) for OpCode view and *Random Forest* for ByteCode view are selected classifiers to obtain positive and negative memberships according to Figure 8.

After training selected classifiers on each view, the evaluation metrics are reported in Table 1. By using the outcomes of these trained classifiers to construct the proposed classifier, performance will be increased as illustrated in Figure 9 The high *Accuracy*, *F-measure*, *AUC*, *MCC* and low false positive rate of the proposed method in Table 1 indicate the robustness of the proposed model to detect malware in compare with single view systems.

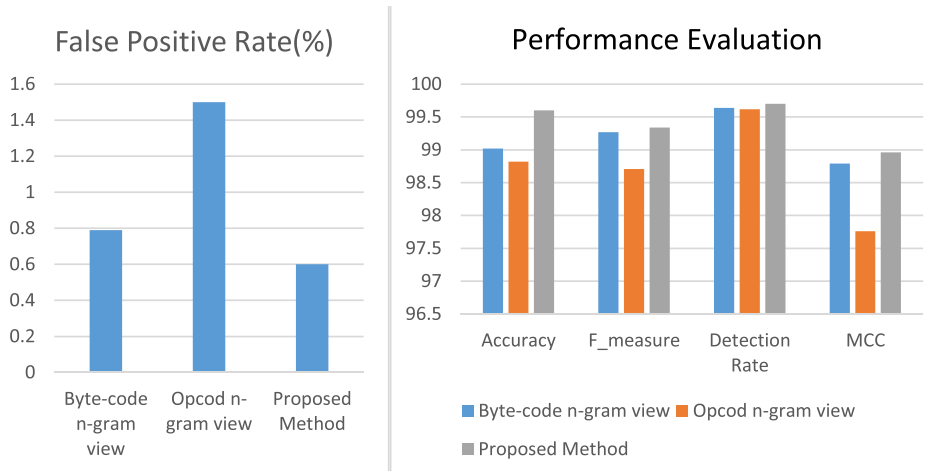
Our proposed method also outperforms Haddadpajuh et al. [14] approach which utilizes Deep Recurrent Neural Network in terms of *Accuracy* (shown in Table 2). It is obvious that employing ensemble multi-view method for malware detection leads to a more accurate detection model.

According to Figure 9 and Table 2, the classifier on ByteCode view has resulted in higher performance in compare with other views. Therefore, when we calculate each view’s weights, the ByteCode view should have greater weights than the other view. As can be seen in Figure 10, both weights of ByteCode view are greater than weights of other views. The reason for the ByteCode view’s greater impact is that the ByteCodes of a file contain header and instruction codes, while OpCode are reflecting instructions only.

The second dataset to evaluate the proposed method is  $D_{PE}$  with three views. In order to obtain positive and negative memberships of each sample, we selected *KNN* (K=7) classifier

**Table 1** Performance evaluation of detection methods on IoT dataset

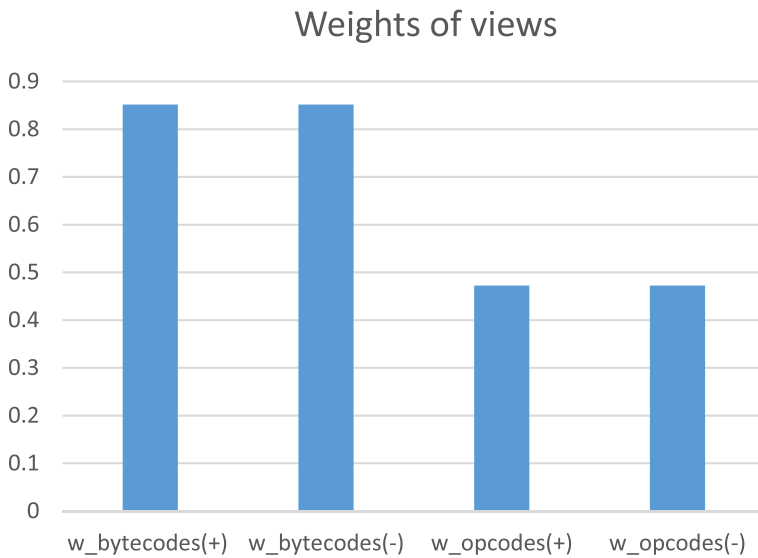
	Accuracy	F-score	TPR	MCC	FPR
ByteCode view	99.02	99.27	99.64	98.79	0.72
OpCode view	99.12	98.71	99.62	97.76	1.5
Proposed model	<b>99.6</b>	<b>99.34</b>	<b>99.7</b>	<b>98.96</b>	<b>0.6</b>



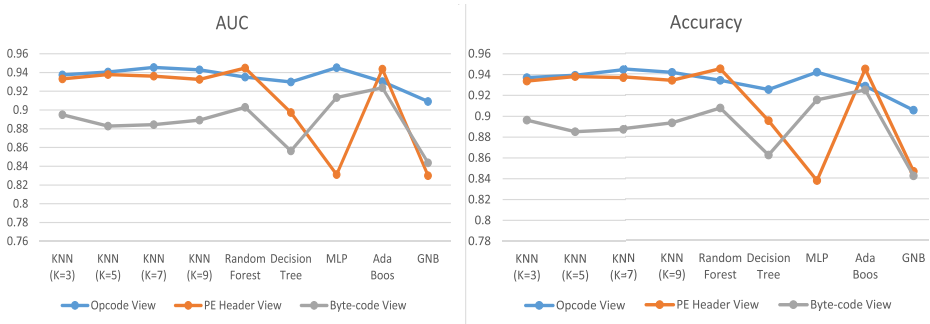
**Figure 9** Performance of proposed method and single views on IoT dataset

**Table 2** Comparison Between the Proposed Method and HaddadPajouh et al. [14]

	Accuracy
Proposed Method	<b>99.6</b>
Haddadpajouh	98.18



**Figure 10** Assigned weights to views of IoT dataset



**Figure 11** Accuracy and AUC of trained classifiers on each view of PE dataset

for OpCode view, *Random Forest* for PE header view and *Ada-boost* classifier for ByteCode view among evaluated classifiers in Figure 11.

Table 3 gives information about how the proposed method improves the performance regarding various metrics against single views. The proposed method achieved high accuracy (98.01%) with a low false positive rate of 1.23%. According to Figure 12, it is clear that our proposed method achieved higher performance in compare with single view systems.

We investigated the weights assigned to the views to ensured that those with higher performance gain more weights and vice versa. As it is demonstrated in Figure 12 and Table 3, the ByteCode view is the worst with the least weight for this dataset. Figure 13 displays the weights assigned to different views and as we expected the weights which are assigned to the ByteCode was less because this dataset contains packed malware (as referred in Section 2) and Bytecodes are worst features for detecting packed malware samples.

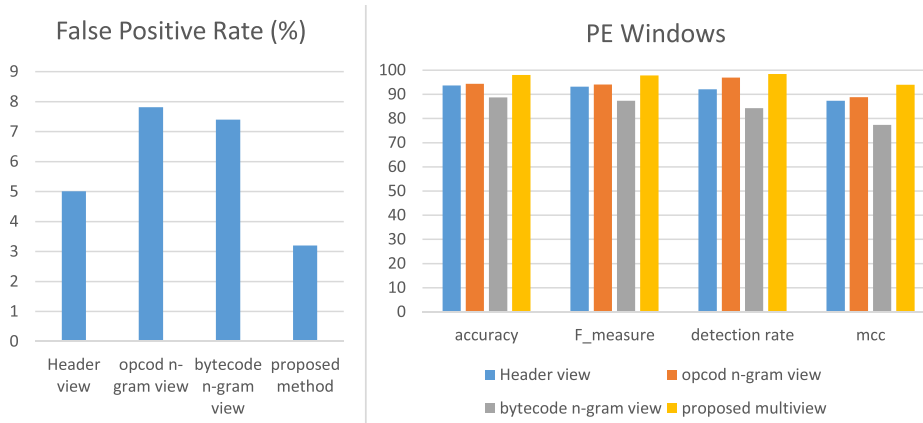
Table 4 gives a comparison between the proposed method and those suggested by Farokhmanesh et al. [10] and Azmoodeh et al. [2] which demonstrates higher performance of our model.

It could be claimed that the proposed model is more robust in compare with previous models as it automatically assigns lower weights to less reliable views while boosting effect of better views by assigning higher weights. To demonstrate this conjecture, we removed the ByteCode view and train our model using remaining two views and as can be seen in Table 5 the performance of the model is not affected. This shows that the original model was mainly trained using views with higher weights and disregarded poor views automatically.

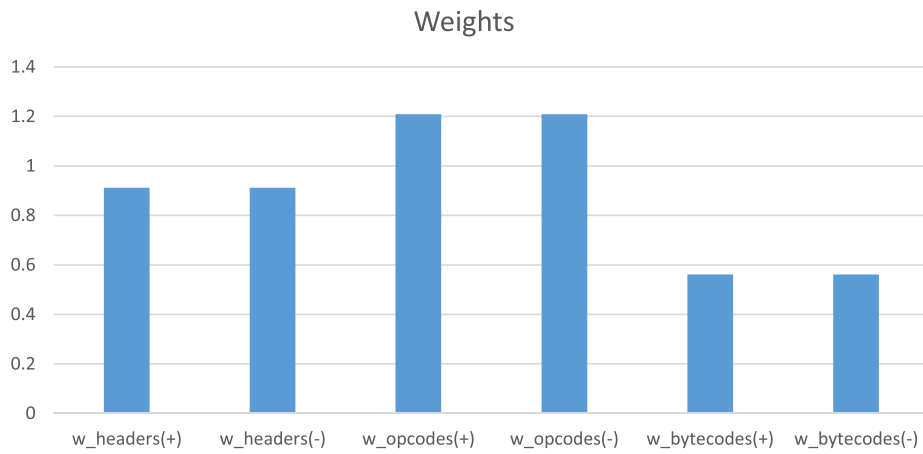
We have tested the proposed method on *Drebin* dataset, which is a very large-scale and unbalanced Android malware dataset to show the performance of our method in detecting malware in unbalanced datasets. Arp et al. [1] used 10 times 3 folds for evaluation and these folds are publicly available. So we utilized these folds to be able to make a fair comparison between the proposed method and the Arp et al. method. Permissions, API calls,

**Table 3** Performance evaluation of detection method on the PE dataset

	Accuracy	F-score	TPR	MCC	FPR
Header view	93.7	93.14	92.14	87.34	5.01
opcode view	94.37	94.1	96.9	88.86	7.81
bytecode view	88.74	87.4	84.28	77.42	7.4
Proposed model	<b>98.01</b>	<b>97.8</b>	<b>97.14</b>	<b>96.01</b>	<b>1.23</b>



**Figure 12** Performance of the proposed method against single views on the PE dataset



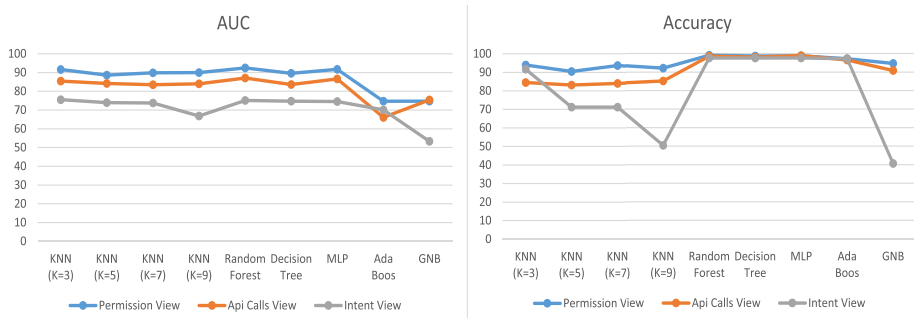
**Figure 13** Assigned weights to views of PE dataset

**Table 4** Performance evaluation of detection methods on PE dataset

	Accuracy	TPR	MCC	FPR
Farokhmanesh et al.	93.15	86.23	92.38	6.55
Azmooddeh et al.	92.16	85.10	98.32	13.41
Proposed model	<b>98.01</b>	<b>96.01</b>	<b>97.14</b>	<b>1.23</b>

**Table 5** Robustness test on PE dataset

	Accuracy	F-score	TPR	MCC
Header view	93.7	93.14	92.14	87.34
opcod n-gram view	94.37	94.1	96.9	88.86
Proposed model	<b>98.01</b>	<b>97.8</b>	<b>98.38</b>	<b>94.01</b>



**Figure 14** Accuracy and AUC of trained classifiers on each view of Drebin dataset

and intents are three views and similar to the previous experiments in this section, we first explored performance of each view with different classifiers. According to Figure 14, the *Random Forest* classifier for permission and intent views and *MLP* classifier for API calls demonstrate highest performance. According to Table 6, our proposed method has a greater performance than trained classifiers on single views (see Figure 15). As can be seen from Figure 16, the Permission view and API calls view have obtained higher weights in compare with Intents view which indicates the relative importance of these two views in detecting Android malware.

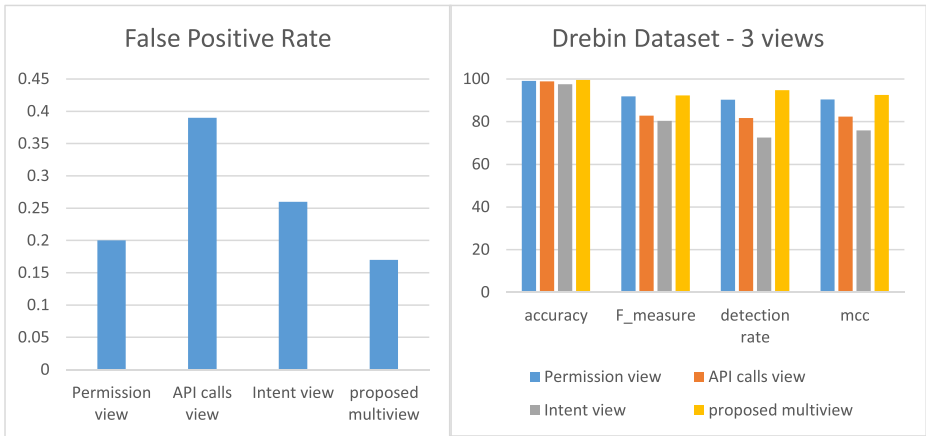
As observed in Table 7, the proposed method has a higher accuracy rate but lower false positive rate in comparison to those of Arp et al. [1]. To evaluate the robustness of the proposed method against previous models, we removed the Intent view and trained our model again with two remaining views. As shown in Table 8, the performance of the model is not affected significantly.

### 5 Related work

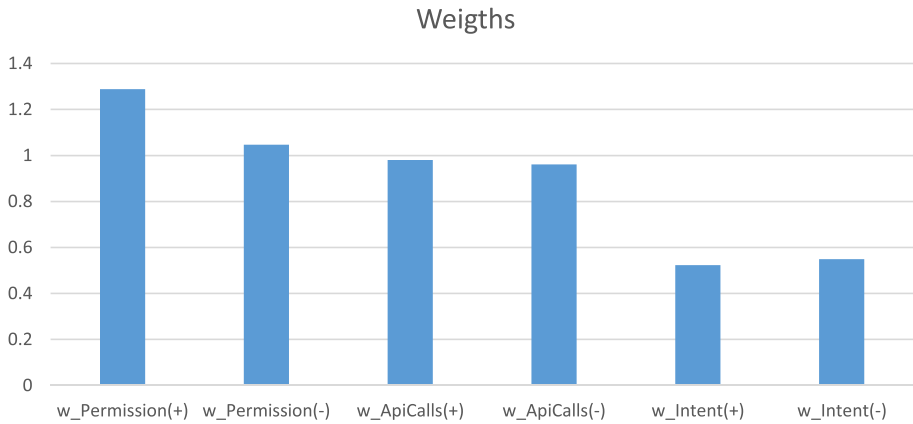
Multi-view learning is tightly connected to ensemble learning [45], where the latter can be used to leverage and merge decisions of a set of classifiers to obtain better predictive performance (relative to using any of the constituent classifiers alone) [28, 46]. Idrees et al. [18], for example, used permissions and intents in a single view classifier to identify Android malware. After extracting permissions and intents, the existence of each item is explored in every sample to build the binary feature vectors. They then applied Boosting, Bagging and Stacking algorithms to evaluate the performance and reportedly achieved 98% of detection rate, 2% of false positive rate with marginal detection time reduction, in comparison to known non-ensemble algorithms (e.g. Decision Table, MLP, and Random Forest). In a separate work, Sheen et al. [36] extracted features from the PE header and

**Table 6** Performance evaluation of detection method on the Drebin dataset

	Accuracy	F-score	TPR	MCC	FPR
Permission view	99.07	91.81	90.26	90.41	0.2
API call view	98.86	82.88	85.74	82.36	0.39
Intent view	97.58	80.35	72.51	75.9	0.26
Proposed model	<b>99.6</b>	<b>92.35</b>	<b>94.8</b>	<b>92.52</b>	<b>0.17</b>



**Figure 15** Performance of proposed method against single views on Drebin dataset



**Figure 16** Assigned weights to views of the Drebin dataset

**Table 7** Performance evaluation of detection method on the Drebin dataset

	TPR	FPR
Proposed Model	<b>94.8</b>	<b>0.17</b>
Arp et al.	94	1

**Table 8** Robustness test on Adndroid dataset

	Accuracy	F-score	TPR	MCC
Permission view	99.07	91.81	90.26	90.41
API call view	98.86	82.88	85.74	82.36
Proposed model	<b>99.4</b>	<b>92.30</b>	<b>94.6</b>	<b>92.45</b>



API calls, and applied different learning algorithms on the same dataset to construct a set of heterogeneous primary classifiers. Then, two ensemble classifier *HS\_ENSEM\_binary* and *HS\_ENSEM\_weighted* were proposed to select a subset of basic classifiers and combined them. The authors demonstrated that their method achieved better performance than recognized ensemble methods such as bagging and boosting with a 99% of detection rate and 0.1% false alarm.

Chakraborty et al. [7] extracted static and dynamic features from the Drebin and Koodous datasets, which include Android applications. Then, they integrated both supervised and unsupervised algorithms to detect malware families. Sahs and Khan [30] extracted a set of features including user-defined permissions, standard permissions, and control flow graph, and subsequently used an anomaly detection based on the One-Class SVM to detect Android malware. Also, RevealDroid [11] uses four extracted feature-sets from APK android files, namely: sensitive APIs, information flows, Intents, and package-level API information, using the Decision Tree classifier for malware detection. Bai et al. [4] merged the feature sets of ByteCode, OpCode and header in a single view, and various classifiers were trained on this single view. Finally the output of these classifiers were combined using voting, stacking and ensemble selection.

However, the approaches discussed above performed only a simple concatenation on their feature vectors (views). Guo et al. [13] divided API call sequences into seven subsequences, such as network, file IO and other operations, and then used each subsequence to build a classification model. The outputs of these models are combined by using BKS algorithm and the final fusion output will be used to label whether a software is malicious or not. Narayanan et al. [21] utilized multi-view learning based on Multiple Kernel Learning (MKL) to detect Android malware. In this approach, five complementary sets of semantic views of apps are considered as extracted views.

## 6 Conclusion and future works

As our society becomes more reliant on cyberspace and other technologies, having the capability to achieve efficient and effective cybersecurity is crucial, particularly from a national security perspective. Therefore, in this paper we focused on malware detection and cyber threat hunting. We demonstrated the potential of using different feature views in malware detection for different platforms. Specifically, we incorporated several well studied feature sets as distinct views, and proposed a multi-view learning approach to integrate these multi-view features. The proposed method utilizes SVM to weigh the obtained views, in order to improve malware detection rate. We then demonstrated that this approach improves the overall classification performance using different datasets.

While we demonstrated the potential to use application files (malware/benign) and their behavior in multiple perspectives (views) in malware detection. Future research will include weighting the views using other linear classification or employing other multi-view classification models, as well as utilization of multi-view deep learning methods to improve the performance of malware detection.


## References

1. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C.: Drebin: effective and explainable detection of android malware in your pocket. In: Ndss, vol. 14, pp. 23–26 (2014)

2. Azmoodeh, A., Dehghantanha, A., Choo, K.K.R.: Robust malware detection for internet of (battlefield) things devices using deep eigenspace learning. *IEEE Trans. Sustain. Comput.* (2018)
3. Azmoodeh, A., Dehghantanha, A., Conti, M., Choo, K.K.R.: Detecting crypto-ransomware in iot networks based on energy consumption footprint. *J. Ambient. Intell. Humaniz. Comput.*, 1–12 (2017)
4. Bai, J., Wang, J.: Improving malware detection using multi-view ensemble learning. *Secur. Commun. Netw.* **9**(17), 4227–4241 (2016)
5. Beel, J., Gipp, B., Langer, S., Breitingner, C.: Research-paper recommender systems: a literature survey. *Int. J. Digital Libraries* **17**(4), 305–338 (2015). <https://doi.org/10.1007/s00799-015-0156-0>
6. Bishop, C.M. et al.: *Neural Networks for Pattern Recognition*. Oxford University Press, London (1995)
7. Chakraborty, T., Pierazzi, F., Subrahmanian, V.: Ec2: ensemble clustering and classification for predicting android malware families. *IEEE Trans. Dependable Secure Comput.* (1), 1–1 (2017)
8. Cui, H., Zhou, Y., Wang, C., Li, Q., Ren, K.: Towards privacy-preserving malware detection systems for android. In: 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), pp. 545–552 (2018)
9. Darabian, H., Dehghantanha, A., Hashemi, S., Homayoun, S., Choo, K.K.R.: An opcode-based technique for polymorphic internet of things malware detection. *Concurrency and Computation: Practice and Experience*, e5173 (2019)
10. Farrokhanesh, M., Hamzeh, A.: Music classification as a new approach for malware detection. *Journal of Computer Virology and Hacking Techniques*, 1–20 (2018)
11. Garcia, J., Hammad, M., Pedrood, B., Bagheri-Khaligh, A., Malek, S.: Obfuscation-Resilient, Efficient, and Accurate Detection and Family Identification of Android Malware. Department of Computer Science, George Mason University, Tech. Rep (2015)
12. Guo, J., Zhu, W.: Partial multi-view outlier detection based on collective learning. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)
13. Guo, S., Yuan, Q., Lin, F., Wang, F., Ban, T.: A malware detection algorithm based on multi-view fusion. In: International Conference on Neural Information Processing, pp. 259–266. Springer (2010)
14. HaddadPajouh, H., Dehghantanha, A., Khayami, R., Choo, K.K.R.: A deep recurrent neural network based approach for internet of things malware threat hunting. *Futur. Gener. Comput. Syst.* **85**, 88–96 (2018)
15. Hearst, M.A., Dumais, S.T., Osuna, E., Platt, J., Scholkopf, B.: Support vector machines. *IEEE Int. Sys. Appl.* **13**(4), 18–28 (1998)
16. Hopkins, M., Dehghantanha, A.: Exploit kits: the production line of the cybercrime economy? In: 2015 Second International Conference on Information Security and Cyber Forensics (Infosec), pp. 23–27. IEEE (2015)
17. Hu, Q., Zhu, P., Yang, Y., Yu, D.: Large-margin nearest neighbor classifiers via sample weight learning. *Neurocomputing* **74**(4), 656–660 (2011)
18. Idrees, F., Rajarajan, M., Conti, M., Chen, T.M., Rahulamathavan, Y.: Pindroid: a novel android malware detection system using ensemble learning methods. *Comput. Secur.* **68**, 36–46 (2017)
19. Kohavi, R. et al.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: *Ijcai*, vol. 14, pp. 1137–1145. Montreal, Canada (1995)
20. Maiorca, D., Biggio, B., Giacinto, G.: Towards adversarial malware detection: lessons learned from pdf-based attacks. *ACM Computing Surveys (CSUR)* **52**(4), 78 (2019)
21. Narayanan, A., Chandramohan, M., Chen, L., Liu, Y.: A multi-view context-aware approach to android malware detection and malicious code localization. *Empir. Softw. Eng.* **23**(3), 1222–1274 (2018)
22. Narayanan, A., Soh, C., Chen, L., Liu, Y., Wang, L.: Apk2vec: semi-supervised multi-view representation learning for profiling android applications. In: 2018 IEEE International Conference on Data Mining (ICDM), pp. 357–366 (2018)
23. Nari, S., Ghorbani, A.A.: Automated malware classification based on network behavior. In: 2013 International Conference on Computing, Networking and Communications (ICNC), pp. 642–647. IEEE (2013)
24. Nguyen-Vu, L., Ahn, J., Jung, S.: Android fragmentation in malware detection. *Comput. Secur.* **87**, 101573 (2019)
25. O’Kane, P., Sezer, S., Carlin, D.: Evolution of ransomware. *IET Netw.* **7**(5), 321–327 (2018)
26. Prayudi, Y., Riadi, I., et al.: Implementation of malware analysis using static and dynamic analysis method. *Int. J. Comput. Appl.* **117**(6) (2015)
27. Raff, E., Sylvester, J., Nicholas, C.: Learning the pe header, malware detection with minimal domain knowledge. In: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, pp. 121–132. ACM (2017)
28. Rokach, L.: Ensemble-based classifiers. *Artif. Intell. Rev.* **33**(1–2), 1–39 (2010)

29. Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., Ahmadi, M.: Microsoft malware classification challenge. arXiv:1802.10135 (2018)
30. Sahs, J., Khan, L.: A machine learning approach to android malware detection. In: 2012 European Intelligence and Security Informatics Conference, pp. 141–147. IEEE (2012)
31. Salehi, Z., Sami, A., Ghiasi, M.: Maar: robust features to detect malicious activity based on api calls, their arguments and return values. *Eng. Appl. Artif. Intel.* **59**, 93–102 (2017)
32. Sami, A., Yadegari, B., Rahimi, H., Peiravian, N., Hashemi, S., Hamze, A.: Malware detection based on mining api calls. In: Proceedings of the 2010 ACM Symposium on Applied Computing, pp. 1020–1025. ACM (2010)
33. Sarma, B.P., Li, N., Gates, C., Potharaju, R., Nita-Rotaru, C., Molloy, I.: Android permissions: a perspective combining risks and benefits Proceedings of the 17th ACM symposium on Access Control Models and Technologies, pp. 13–22. ACM (2012)
34. Santos, I., Devesa, J., Brezo, F., Nieves, J., Bringas, P.G.: Opem: a static-dynamic approach for machine-learning-based malware detection. In: International Joint Conference CISIS' 12-ICEUTE 12-SOCO 12 Special Sessions, pp. 271–280. Springer (2013)
35. Shalaginov, A., Banin, S., Dehghantanha, A., Franke, K.: Machine learning aided static malware analysis: a survey and tutorial. *Cyber Threat Intelligence*, 7–45 (2018)
36. Sheen, S., Anitha, R., Sirisha, P.: Malware detection by pruning of parallel ensembles using harmony search. *Pattern Recogn. Lett.* **34**(14), 1679–1686 (2013)
37. Shijo, P., Salim, A.: Integrated static and dynamic analysis for malware detection. *Procedia Comput. Sci.* **46**, 804–811 (2015)
38. Sikorski, M., Honig, A.: *Practical Malware Analysis* O'Reilly (2012)
39. Singh, A., Dutta, D., Saha, A.: Migan: malware image synthesis using gans. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 10033–10034 (2019)
40. Skolka, P., Staicu, C.A., Pradel, M.: Anything to hide? Studying minified and obfuscated code in the web. In: The World Wide Web Conference, pp. 1735–1746. ACM (2019)
41. Steinwart, I., Christmann, A.: *Support Vector Machines*. Springer, Berlin (2008)
42. Sun, S.: A survey of multi-view machine learning. *Neural Comput. Applic.* **23**(7–8), 2031–2038 (2013)
43. Taheri, M., Azad, H., Ziarati, K., Sanaye, R.: A quadratic margin-based model for weighting fuzzy classification rules inspired by support vector machines. *Iranian J. Fuzzy Sys.* **10**(4), 41–55 (2013)
44. Wang, Q., Guo, W., Zhang, K., Ororbia, I.I., Xing, A.G., Liu, X., Giles, C.L.: Adversary resistant deep neural networks with an application to malware detection. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1145–1153. ACM (2017)
45. Xu, C., Tao, D., Xu, C.: A survey on multi-view learning. arXiv:1304.5634 (2013)
46. Xu, Z., Sun, S.: An algorithm on multi-view adaboost. In: International Conference on Neural Information Processing, pp. 355–362. Springer (2010)
47. Yakura, H., Shinozaki, S., Nishimura, R., Oyama, Y., Sakuma, J.: Neural malware analysis with attention mechanism. *Comput. Secur.* **87**, 101592 (2019)
48. Ye, Y., Hou, S., Chen, L., Lei, J., Wan, W., Wang, J., Xiong, Q., Shao, F.: Out-of-sample node representation learning for heterogeneous graph in real-time android malware detection. In: 28th International Joint Conference on Artificial Intelligence (IJCAI), 2019 (2019)
49. Ye, Y., Li, T., Adjeroh, D., Iyengar, S.S.: A survey on malware detection using data mining techniques. *ACM Computing Surveys (CSUR)* **50**(3), 41 (2017)
50. You, I., Yim, K.: Malware obfuscation techniques: a brief survey. In: 2010 International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA), pp. 297–300. IEEE (2010)
51. Zhao, J., Xie, X., Xu, X., Sun, S.: Multi-view learning overview: recent progress and new challenges. *Information Fusion* **38**, 43–54 (2017)
52. Zhou, D., He, J., Candan, K.S., Davulcu, H.: Muvir: multi-view rare category detection. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)

## Affiliations

**Hamid Darabian<sup>1</sup> · Ali Dehghantanha<sup>2</sup> · Sattar Hashemi<sup>1</sup> · Mohammad Taheri<sup>1</sup> · Amin Azmoodeh<sup>2</sup> · Sajad Homayoun<sup>3</sup> · Kim-Kwang Raymond Choo<sup>4</sup>  · Reza M. Parizi<sup>5</sup>**

Hamid Darabian  
h.darabian@cse.shirazu.ac.ir

Ali Dehghantanha  
adehghan@uoguelph.ca

Mohammad Taheri  
motaheri@shirazu.ac.ir

Amin Azmoodeh  
amin@cybersciencelab.org

Sajad Homayoun  
s.homayoun@sutech.com

Kim-Kwang Raymond Choo  
raymond.choo@fulbrightmail.org

Reza M. Parizi  
rparizi1@kennesaw.edu

<sup>1</sup> Department of Computer Science and Engineering, Shiraz University, Shiraz, Iran

<sup>2</sup> Cyber Science Lab, School of Computer Science, University of Guelph, Ontario, Canada

<sup>3</sup> IT and Computer Engineering Faculty, Shiraz University of Technology, Shiraz, Iran

<sup>4</sup> Department of Information Systems and Cyber Security and Department of Electrical and Computer Engineering, The University of Texas at San Antonio, San Antonio, TX 78249, USA

<sup>5</sup> Department of Software Engineering and Game Development, Kennesaw State University, Marietta, GA 30060, USA