Society for Science and Education
United Kingdom
TMLAI Transactions on Machine Learning and Artificial Intelligence
Volume 2, Issue 3
ISSN: 2054 - 7390

An Android Malware Detection Architecture based on Ensemble Learning

Mehmet Ozdemir, Ibrahim Sogukpinar
Department of Computer Engineering, Gebze Institute of Technology, Gebze, Kocaeli, Turkey;
mehmet.ozdemir@tubitak.gov.tr, ispinar@bilmuh.gyte.edu.tr

ABSTRACT

In the scope of anomaly based Android malware detection, different type of features has been used to represent applications and lots of algorithms have been applied to evaluate these features. Although researchers have reported accurate results, in order to improve accuracy, sensitivity and generalization, we suggest using an ensemble learning approach for Android malware detection. In this study, we propose to use an ensemble learning system whose base learners are built with different feature subsets which are extracted and processed with multiple methods, and selected with a proposed selective ensemble approach which is based on three criteria: Accuracy, sensitivity and diversity.

Keywords: Ensemble Learning, Multiple Classifier Systems, Mixture of Experts, Selective Ensemble, Malware Detection, Android Malware.

1 INTRODUCTIN

Detection of malware using data mining techniques requires representing applications as features. These features are then used to build mining models which provide predictions about maliciousness of applications. However, selecting most suitable features among lots of feature types and processing the selected features with correct algorithms is not a simple task.

In order to represent applications completely, selected feature types must be carefully decided. Choosing few features types may not represent all applications sufficiently. For instance, network activity of an application can be recorded by executing that application and can be used as features in order to represent that application. However, it can't be guaranteed that these features will enough to distinguish malware and benign applications because there is a good chance of being malware without network activity. On the other hand, feature types which are complement of each other must be considered. For example, if API calls are used as

DOI: 10.14738/tmlai.23.261
Publication Date: 9th June 2014
URL: http://dx.doi.org/10.14738/tmlai.23.261

features, these calls must be gathered from both native code and byte code because it isn't known where the malicious intent would be in advance.

Analysis type of extracted features may be another complementing factor of features because static and dynamic analyses are generally considered as complement of each other. For instance, it is hard to gather useful information from obfuscated code via static analysis but dynamic analysis can reveal that code's behavior. On the other hand, dynamic analysis may not reveal some malware intentions (e.g. time bomb attacks) because of limited execution time but static analysis may catch some pattern about these kinds of intentions.

Choosing the mining algorithms are also important as well as chosen feature types. Once the features are extracted, feature selection algorithms are generally applied before providing these features to the learning algorithms. There are kinds of feature selection and learning algorithms whose assumptions are different from each other. None of these algorithms have been proven to be best for a specific problem and thus, several methods should be compared before making predictions. Additionally, comparison of chosen algorithms may be crucial. For example, two different learning models may have the same accuracy but their predictions may be totally different.

Contribution of this paper is twofold. First one is to show benefits of ensemble learning for malware detection problem. Second one is to increase accuracy and sensitivity of malware detection operation with proposed architecture. In order to do this, we propose to use an ensemble learning system whose base learners are generated using different kind of features which are extracted from different aspects of applications, and processed using different kind of selection and learning algorithms. After that, some of the generated base learners are selected with a proposed heuristic algorithm to improve accuracy and sensitivity. Finally, selected base learners are combined via stacking or majority voting to build an ensemble system.

The rest of the paper is organized as follows. Related works are described in Section 2. Overview of ensemble learning and feature selection is presented in Section 3. Architecture of proposed method is explained in Section 4. Experimental results and evaluation are given in Section 5. Conclusions and future work are described in the last section.

2 RELATED WORKS

Misuse and anomaly detection are two general approaches to reveal malware applications. In misuse detection, signatures are generated for each specific kind of malware and once a signature is generated, that specific malware can be found precisely [10]. However, this method fails to detect novel malware. On the other hand, anomaly based systems build a general model using an application dataset and applications that don't fit to this model are considered as anomalous [23]. Although novel malware can be detected by anomaly based systems, false predictions may be problem for this method.

Features are extracted from applications by applying two general approaches: Dynamic and static analysis. In dynamic analysis, applications are run on a device or emulator and behavior of the application or system is watched. System calls [4], sensitive data tracking [9], system logs [22], messaging & call information, CPU load etc. [23] are some dynamically extracted features. In static analysis, features are extracted without running the application. API calls [32], opcodes and operands [30], class hierarchies, used packages [32] and control flow graphs of applications [20] are some statically extracted features from compiled source of applications. Also it is possible to extract static features from Android application package (apk) content such as resource locations and Android permissions [32]. Also, meta information such as description, download count and price of application [26] can be statically extracted from marketplace.

Zhou and Jiang [31] collected 1260 malware samples and used these samples to test existing mobile security tools. Best accuracy rate was reported as 79.6%. After this study, researchers have reported better accuracies from their studies. However, sensitivities of some studies were considerably low. Sensitivity[1] is the measure of ability to identify positive (malware) samples correctly. It is very important for malware detection tools because labeling a malware application as benign (false negative) will cause malware application to be added to the application market, which is not acceptable, while false positives can be fixed by security experts of the application market. Hence, this situation motivated us to create a detection tool both accurate and sensitive.

3 OVERVIEW

3.1 Ensemble Learning

Ensemble learning[2] is a machine learning method which is used to improve accuracy of learning systems by *generating* a set of base learners and then *combining* outputs of these base learners. Effectiveness of ensemble learning has been proven in several studies [7] when the base learners have error rates less than random guessing and their errors are uncorrelated (i.e. base learners are diverse).

There are three stages for constructing ensemble systems [25]. First one is to generate accurate and diverse base learners. Diversity is a term which is used to define variation among outputs of learning models. It is an important factor for effective ensemble systems because if base learners are identical, obviously, combining them wouldn't provide any information gain.

Although the importance of diversity over effective ensemble systems is clear, there is not a common explanation for diversity in literature. Though, there are several investigations to explain diversity among learning models for regression and classification problems [3].

[1] Recall rate, true positive rate and hit rate are other terms used for sensitivity.

[2] This term is also known as multiple classifier systems, mixture of experts, committees of learners.

Diversity measures are one of the proposed methods to measure diversities among base learners and it fall into two categories, the pair wise measures and the non-pair wise measures. In the pair wise measures, firstly, diversity is measured for each pair of base learners and then diversity of the ensemble system is calculated by averaging these pair wise diversities. In the non-pair wise measures, diversity of the ensemble is measured directly without considering divergence between pair of base learners. In order to construct better ensemble systems, Kuncheva and Whitaker [16] compared several diversity measures for classification problems in terms of correct/incorrect outputs. As a result, they reported that existing divergence generation methods are valid but measuring diversity in order to build better ensemble systems is an open problem.

Since there is not a general acceptance about diversity, diverse base learners are tried to be generated intuitively with different methods including, using different learning algorithms [15], using different parameters of the same learning algorithm [13], using different subsamples in the training set [11], using different feature subsets of the training set [14], manipulating the output targets [6], injecting randomness into algorithms [17].

Selecting appropriate base learners is the second stage to construct ensemble systems. Instead of using all generated base learners, optionally, a subset of them is selected. Investigations show that such a stage improves the performance of the ensemble system [19].

There are two strategies for the concept of base learner selection as the "direct strategy[3]" and the "overproduce and choose strategy[4]". Direct selection of base learners is done internally by learning algorithm itself at the training phase such as selecting base learners with pruning functions in boosting. In overproduce and choose strategy, firstly, lots of base classifiers are generated and then a subset of them is selected by applying search algorithms or rules. These rules or search algorithms evaluate base learners based on one or more evaluation criteria like accuracy or diversity measure. Selecting best ones [18], selecting via heuristics [24], selecting with genetic algorithms [21] are some selection techniques in literature.

Final stage of constructing ensemble systems is to combine selected base learners. In order to combine base learners' outputs for final decision, a simple majority voting may be used (or a simple averaging for regression). Or, another learning model[5] may be constructed using base learners' outputs (i.e. stacking [28]). Actually, many combination schemes have been proposed which are based on different theories. Studies that convert class labels to continuous values and apply regression to the converted values for combining classifier outputs even exist. More

[3] This strategy is also known as dynamic selection, or, test and select methodology.

[4] Static selection or selective ensemble terms are also used for this strategy.

[5] This learning model is called as meta or strong learner.

details can be found Tulyakov et al.'s study which gives a comprehensive review of combination schemes for classification problems under different categorizations [27].

In conclusion, main purpose of ensemble learning is to improve generalization of learning models in general, not to find best accuracy for a specific dataset. Hence, accuracy of an ensemble system may be lower than its best base learner but commonly are expected to be higher than average. Though, ensemble systems' accuracies may be as high as their best base learner's. For instance, Saso and Bernard constructed an ensemble system using stacking scheme and they compared performance of their ensemble system with a best classifier chosen from a cross validation. They reported that results were comparable [8].

3.2 Feature Selection

Representing applications as features may produce very high dimensional data. Number of the features may be as high as tens of thousands. Running learning algorithms with this data would increase time and space complexity. Instead of taking into account all of the produced features; a subset of them is selected using a feature selection algorithm. Several studies showed that, using such an algorithm may remove irrelevant and redundant feature and also increase accuracy [2].

Guyon and Elisseeff presented a detailed review about feature selection [12]. Although feature selection for unsupervised learning was mentioned, main focus of their study is supervised learning. In the following part, feature selection for supervised learning will be discussed briefly.

There are two general approaches for feature selection algorithms: The *wrapper* approach and the *filter* approach [29]. The wrapper approach evaluates different feature subsets' usefulness testing them on a specific learning algorithm. In a typical wrapper approach, a feature subset is selected via a search algorithm and learning models are generated with selected features by applying a cross validation, then accuracies of these models are measured. After these operations, a new feature subset is selected by the search algorithm and previous steps are repeated until the search algorithm terminates. Finally, features that have been used to generate most accurate model are treated as selected features.

Different search algorithms can be used to traverse feature space including, exhaustive search, greedy forward/backward search or genetic algorithms. Although exhaustive search can find the optimum feature subset, other algorithms are generally preferred because multiple models must be generated within a cross validation for each selected feature subset, and this takes a large amount of time. If the number of features is very high, this approach would be infeasible regardless of the selected search method. On the other hand, this approach is multivariate which means presence of different features together is considered. This may be important because some variables that are useless individually may be useful with other features [12].

In the filter approach, features are selected by performing some statistical analysis on features and labels. This approach doesn’t require generating learning models as in wrappers and thus filters work faster than wrappers in general. Although it is possible to filter features either individually (univariate[6]) or in subsets (multivariate), most of the filters are univariate. Additionally, selected features by filters can be used on different learning algorithms.

In addition to the filter and the wrapper approaches, some authors mention another method, the *embedded* approach. In this approach, selection operation is done at the training phase by an internal function of learning algorithm (e.g. decision trees). This approach may seem like the wrapper approach at first glance since selection operation is done on a specific learning algorithm. In fact, they are different because embedded approach doesn't include cross validation and repeated learning model generation for different feature subsets as in wrappers.

4 DETECTION ARCHITECTURE

In this section, we present detection architecture in order to make predictions about maliciousness of applications with high accuracy and high sensitivity. As discussed in previous sections, instead of evaluating one aspect of applications with one algorithm, considering different aspects with multiple algorithms may be useful. Discussed architecture has been designed to serve such a purpose. Discussed architecture is component based and each component represents a different type of evaluation in the problem. For example, feature extraction/preprocessing/selection and base learner generation/selection/combination operations can be done using different methods so that each instance of these methods are implemented in different components. For example, static API call and static opcode are two instances of feature extraction components.

There is a dependency among components of discussed architecture. Outputs of some components may be inputs of other components. For example, base learner generation components uses outputs of feature selection components and provides inputs for base learner selection components. Steps to create an ensemble system for proposed architecture are presented in Figure 1. Each step represents a component of the system and dependencies of components can be observed from this Figure.

[6] Also known as feature ranking or weighting.

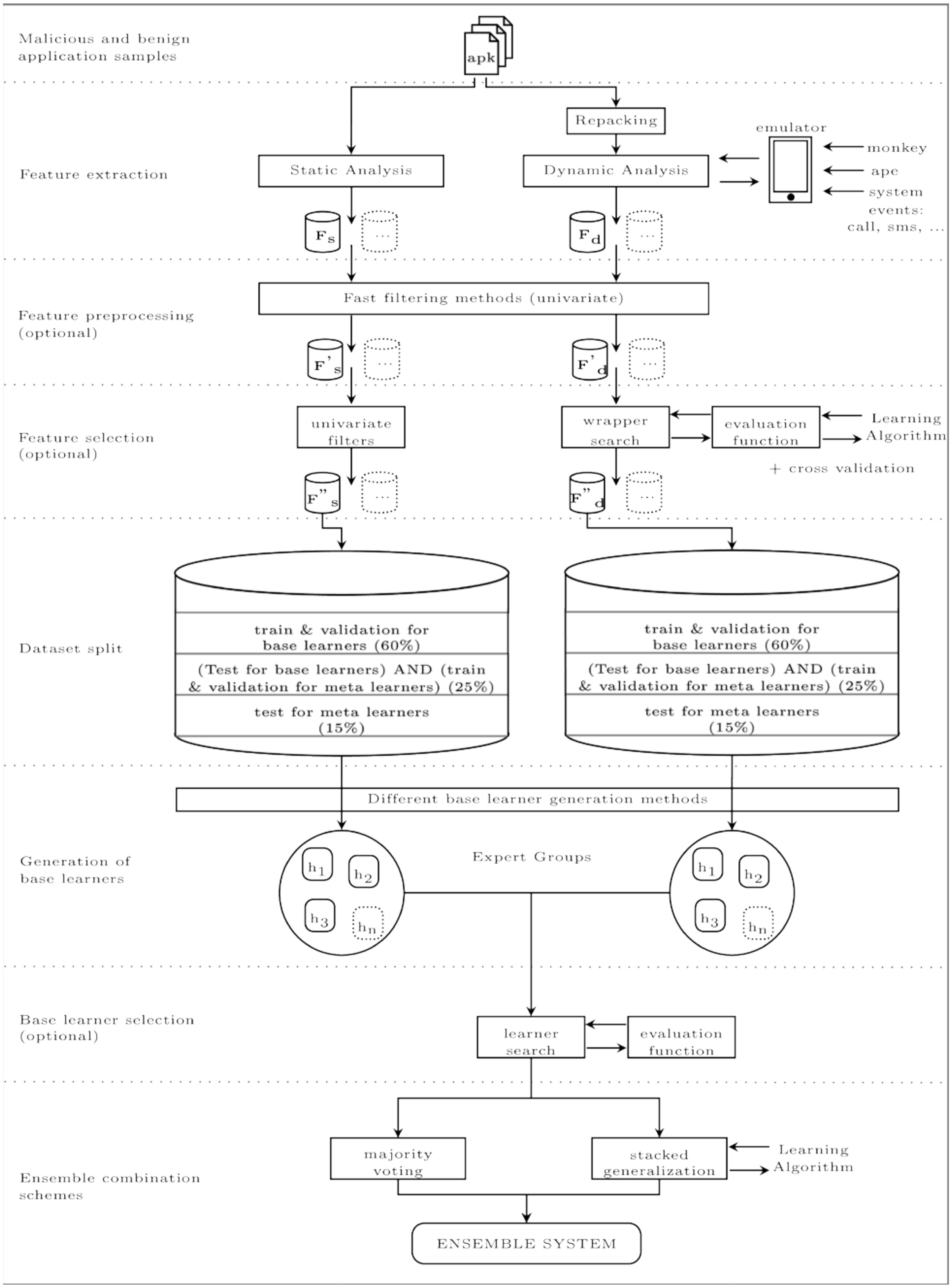

Malicious and benign application samples
apk
Feature extraction
Static Analysis
Repacking
Dynamic Analysis
emulator
monkey
ape
system events: call, sms, ...
Fs
Fd
Feature preprocessing (optional)
Fast filtering methods (univariate)
F's
F'd
Feature selection (optional)
univariate filters
wrapper search
evaluation function
Learning Algorithm
+ cross validation
F''s
F''d
Dataset split
train & validation for base learners (60%)
(Test for base learners) AND (train & validation for meta learners) (25%)
test for meta learners (15%)
Different base learner generation methods
Generation of base learners
Expert Groups
h1
h2
h3
hn
Base learner selection (optional)
learner search
evaluation function
Ensemble combination schemes
majority voting
stacked generalization
Learning Algorithm
ENSEMBLE SYSTEM

Figure 1: Android malware detection architecture.

As seen in Figure 1, it is possible to extract features via static and dynamic analysis. Although applications can be analyzed directly in static analysis, some preparation may be required before dynamic analysis. In this study, API calls of byte code were collected by repacking and hooking the API calls in byte code. API calls of native code were collected by using a trick on Linux library loader (i.e. LD_PRELOAD trick).

Dynamic analysis is made on an emulator by sending random events via *Android monkey* and also a tool is developed (*ape)* which sends fixed predetermined events such as touch and key events. Additionally, some system events are sent like incoming call, sms and geo location change events during dynamic analysis to reveal malicious intents.

An important point to consider for feature extraction process in Figure 1 is presence of multiple datasets. Assume that, an application set is being analyzed with two static feature extraction components: *native API calls* and *Dalvik Byte API calls*. Number of samples in outputs of these two components may not be same because all Android applications have to contain Dalvik byte code but native code is optional. Hence, sample count for native API call will be smaller. Similar situation is valid for dynamic analysis. It can't be possible to repack and run all applications on emulator, or even if they can be run, enough information may not be gathered.

If the number of extracted features per application is very high, a preprocessing operation can be used to decrease feature size to an acceptable value. Although only univariate filters are displayed in Figure 1 for preprocessing step, it is also possible to use multivariate filters.

Feature selection operations may be used to select most informative features or to increase efficiency and effectiveness.

Feature extraction and selection operations prepare the datasets. In order to build learning models from these datasets, samples for training and testing learners must be specified. For this purpose, datasets are divided into three parts. First part is used to train and validate base learners (60%). Second part is used to test base learners (25%). It is also used for training/validating the meta learners when stacked generalization is used as combination scheme. And third part is used to test meta learners (15%) which are generated using second part's data. If the majority voting is chosen as combination scheme, only third part is used for testing.

Base learners are also known as *experts* or *weak learners*. However, they can be as strong as they can. For example, in order improve generalization, a cross validation is often applied and it is a kind of ensemble method. So that weak learners aren't actually weak. Also, it is possible to use another ensemble learning algorithm (e.g. adaboost) to create base learners as well.

There can be different datasets in the system as discussed previously. Base learners which are generated using the same dataset are called in the same expert group. For instance, base learners which are generated using the dynamic native API call dataset are called as *native API call experts* and their group is called as *native API call expert group*.

Selection of base learners is an optional operation and all base learners may be chosen directly instead of selecting some of them. However, if there are too many learners in the system, selection may improve time efficiency, accuracy etc. Depending on number of learners, different search algorithms and evaluation functions can be used.

Finally, in order to construct ensemble systems, selected base learners are combined. Only majority voting and stacking displayed in Figure 1 but other techniques can be used too.

5 EXPERIMENTAL RESULTS AND EVALUATION

Proposed solution in this study is based on supervised learning. In general, a dataset D is given, which contains samples of the form $D = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m)\}$ where $\mathbf{x}_\mathrm{i}$ values are vectors like $< x_1, x_2, \ldots, x_n >$, and y_i values are the set of possible labels $y = \{benign, malicious\}$. A hypothesis [7] $h: \boldsymbol{x} \rightarrow y$ is generated using a learning algorithm L and the samples in D. Then, generated hypothesis is used to identify new applications whether they conform to a specific class.

In order to construct an ensemble system, a set of hypotheses H will be generated using a set of datasets $\boldsymbol{D}$ and a set of learning algorithms L. Steps to create these sets are described in the following sections.

5.1 Data Representation

1225 malware samples were used in this study which was collected by Zhou and Jiang [31]. Additionally, 1225 popular applications from different categories were downloaded from *Google Play* via *Google Play Crawler* [5] and these applications considered as benign.

In order to provide a complete evaluation, 4 different feature type were extracted from applications which are believed to complement of each other: Static and Dynamic Native API calls, Static and Dynamic Dalvik Byte API calls.

Static Native API calls to Linux glibc and Android functions were extracted from applications which contain one or more shared objects. Firstly, a vocabulary of used native functions was created ($x = \{log, send, \ldots\}$) and then presence information of these functions per application was recorded as a vector like $x = < 0, 1, \ldots >$. *Static Dalvik Byte API calls* to Java Core and Android methods were also extracted similar way.

Dynamic analysis produces a sequence of API calls like *log, log, send,....* In order to create a vocabulary from sequence of calls, 2-gram representation was used, whose sliding window was incremented by 1 ($x = \{log + log, log + send, \ldots\}$) . Presence information of 2-gram sequences was stored in vectors as in static analysis.

[7] I.e. classifier or learning model.

Although most of the methods for Dalvik Byte code were hooked, only a limited native function was hooked for dynamic analysis because hooking all native functions brought an enormous burden to the system since these calls are watched for a system process named *Zygote*[8]. Hence, a limited native functions were chosen by considering static analysis results (write, getuid, getenv, dup, etc.).

As discussed before, the number of samples in different datasets may not be same (e.g. each application doesn't have to contain a native code). Sample count of each created database and the number of features within these datasets is presented in Figure 2. This figure gives a summary of feature extraction, feature filtering, feature selecting and base learner generation steps of proposed architecture. For instance, after the static native (sn) analysis, a dataset was produced which contains 988 samples and 1.147 features. These features were processed with two filtering algorithms and feature size was reduced to 1000. After that, feature selection operations were applied to the filtered features. Information gain and chi square filters were also used as selection algorithms and top 50 features were selected. Additionally, a wrapper approach was applied for two filtered datasets, using forward search as search algorithm and CART as learning algorithm. 3 and 4 features were selected by the wrapper approach based on the error rates for information gain and chi square datasets respectively. Finally, a set of static native dataset was created $D = \{D_{SN1}, D_{SN2}, D_{SN3}, D_{SN4}\}$.

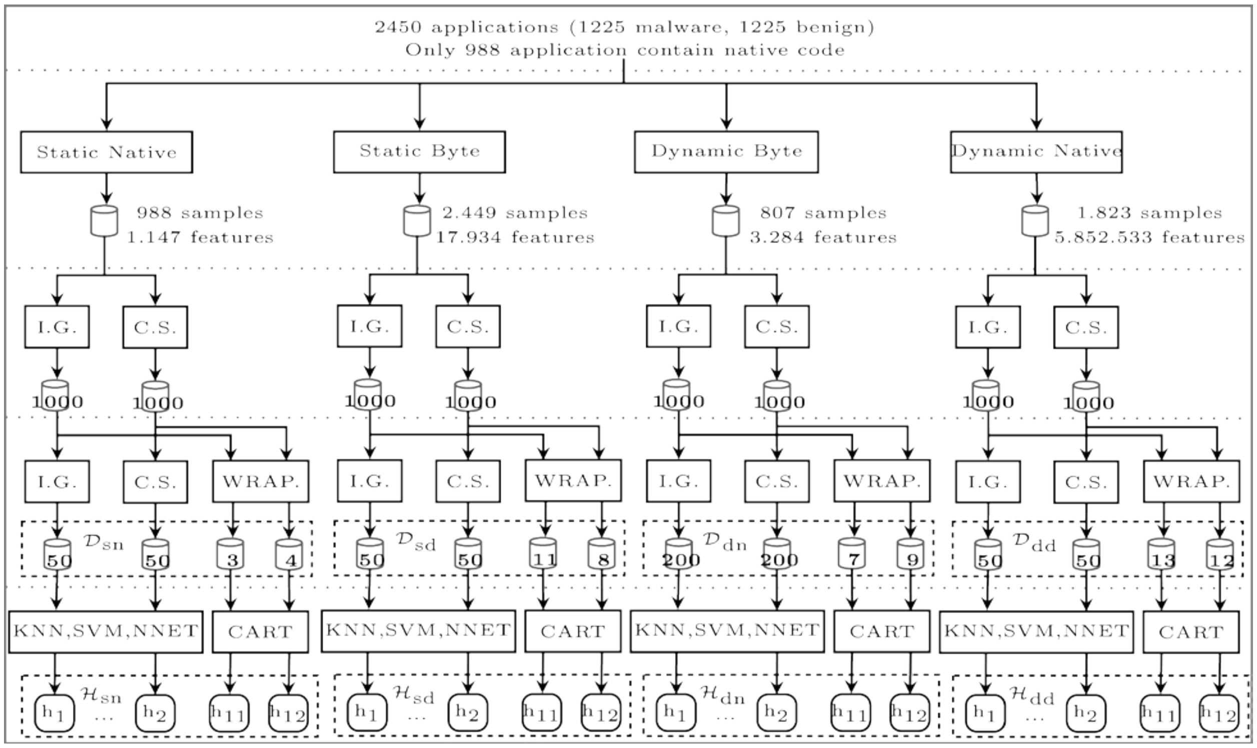


Figure 2: Data representation and base learner generation overview.

[8] Zygote manages all system applications by forking child processes for each one.

Similar steps were applied for the other analysis types. Although most of the steps are identical, number of selected features in dynamic dalvik (dd) analysis was 200 because when 50 features were selected divergence of learning models was very low. At the end, a set of datasets was created which contains all type of dataset sets $\boldsymbol{D} = \{D_{sn}, D_{sd}, D_{dn}, D_{dd}\}$.

5.2 Generating Base Learners

Base learners were generated using the learning algorithms k-NN, NNet, SVMLinear, SVMPoly, SVMRadial and CART. k-NN, NNet and SVM algorithms were used to generate base learners from datasets which were created via filters. On the other hand, CART algorithm was used for datasets that were created using wrapper approach (Fig. 2). General steps for creating base learners are given in Algorithm 1.

Algorithm 1: Steps to generate base learners.

```
Require:
    Datasets for each analysis type: $\boldsymbol{\mathcal{D}} = \{\mathcal{D}_{sn}, \mathcal{D}_{sd}, \mathcal{D}_{dn}, \mathcal{D}_{dd}\}$
    Subsets of each dataset: $\mathcal{D}_i = \{D_{i_1}, D_{i_2}, D_{i_3}, D_{i_4}\}$ where $i = \{sn, sd, dn, dd\}$
    Learning algorithms: $\mathcal{L} = \{L_{knn}, L_{nnet}, L_{svm}, L_{cart}\}$
Ensure:
    A set of all hypothesis: $\boldsymbol{\mathcal{H}} = \{\mathcal{H}_{sn}, \mathcal{H}_{sd}, \mathcal{H}_{dn}, \mathcal{H}_{dd}\}$
    Hypothesis per analysis types: $\mathcal{H}_i = \{H_{i_1}, H_{i_2}, H_{i_3}, H_{i_4}\}$ where $i = \{sn, sd, dn, dd\}$
1: $\boldsymbol{\mathcal{H}} \leftarrow \{\}$
2: for all $\mathcal{D} \in \boldsymbol{\mathcal{D}}$ do
3:     $\mathcal{H} \leftarrow \{\}$
4:     for all $D \in \mathcal{D}$ do
5:         $H \leftarrow$ GENERATEHYPOTHESIS$(D, L, 5, 5)$
6:         Add $H$ to $\mathcal{H}$
7:     end for
8:     Add $\mathcal{H}$ to $\boldsymbol{\mathcal{H}}$
9: end for
   where
1: procedure GENERATEHYPOTHESIS($D$, $L$, k=5, tune=5)
2:     $D_{train}, D_{test} \leftarrow$ SPLIT$(D)$ # Split data set to train and test sets
3:     $H \leftarrow$ TRAIN$(D_{train}, L, k = 5, tune = 5)$ # Train a hypothesis
4:     $ACC, SNS \leftarrow$ PREDICT$(H, D_{test})$ # Store ACC and SNS from prediction
5:     return $H, ACC, SNS$
6: end procedure
```

An important point in Algorithm 1 is the parameters of *Train* function. "k" parameter is used in k-fold cross validation. And, "tune" parameter is used to try different meta parameter values for the learning algorithm L. For example, default k value for k-NN is 5 and when this algorithm is used with a tune parameter value 5, $k = \{5, 7, 9, 11, 13\}$ values will be tried for each fold.

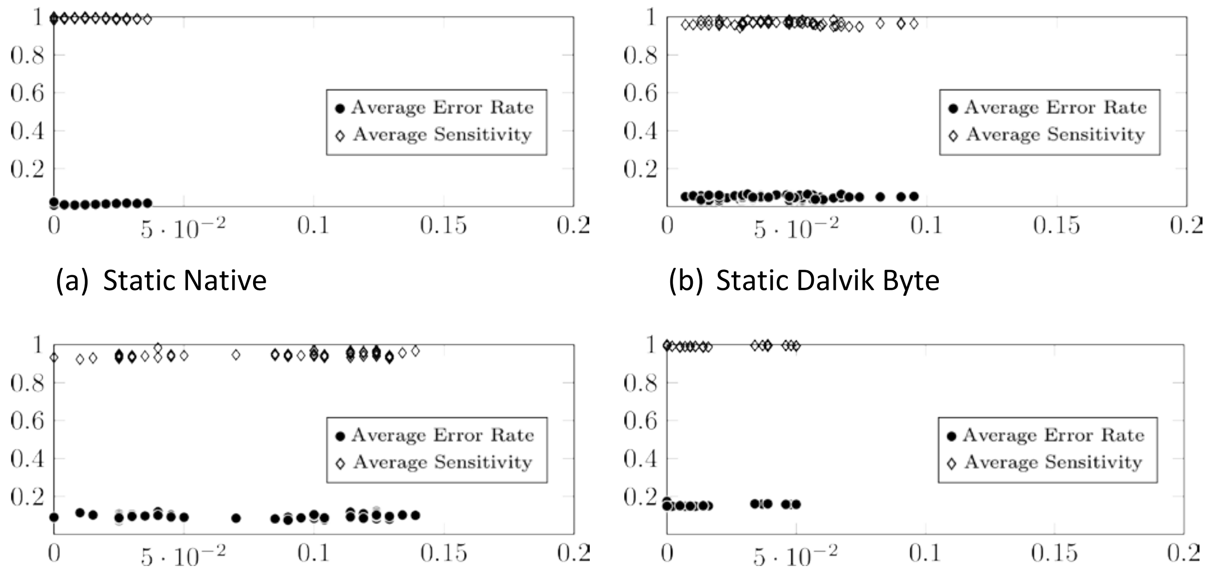
After generating base learners, diversities between each base learner pair were calculated using *disagreement measure* with the formula in Equation 1 assuming K and M are two classifiers, and *a* is the number of samples labeled as malicious by K while they were labeled as benign by M, and *b* is the number of samples labeled as benign by K while they were labeled as malicious by M, *S* is the total sample count. Disagreement measure values which close to 0 means less divergent pairs.

$$dis_{KM} = \frac{a + b}{S} \tag{1}$$

Disagreement measure is a pair wise measurement and it is possible to display produced diversities between base learner pairs. Figure 3 presents diversity-error rate and diversity-sensitivity diagrams using the Equations 2 and 3 for error rate and sensitivity respectively. This figure shows that diverse base learner generation approaches were successful (e.g. using different learning algorithms). For example, there are some base learner pairs whose diversity measure is almost 0.1 in the static Dalvik byte code dataset. Considering the number of samples in this dataset (2.449), diversity measurement value 0.1 of a base learner pair means that pairs have 244 different predictions.

$$Error\ Rate = \frac{\mathrm{FP} + \mathrm{FN}}{TP + FP + TN + FN} \tag{2}$$

$$Sensitivity = \frac{TP}{TP + FN} \tag{3}$$



(a) Static Native

(b) Static Dalvik Byte

(c) Dynamic Native

(d) Dynamic Dalvik Byte

Figure 3: Diversity-Error Rate and Diversity-Sensitivity diagrams for each type of dataset. Error rate and sensitivity were denoted by y-axis while diversity was denoted by x-axis.

5.3 Constructing Ensemble Systems

In order to construct ensemble system, three types of combination scheme were applied. First one was a simple majority voting. When number of votes was equal for the majority voting, the final output was produced as *malicious* in order to increase sensitivity. Second scheme was stacking which evaluates all base learners' outputs with multiple learning algorithms. And the third scheme was stacking too but instead of using all base learners, a subset of them was selected before combination using a simple heuristic as in Algorithm 2.

Algorithm 2 proposes a base learner selection approach based on selecting most and least learners considering their accuracy and sensitivity. Since predictions of selected experts will be used to build meta learning models (i.e. stacking), learning the least cases as the most cases would be useful. Hence, base learner pairs were selected using the *FindMost*Pair* and *FindLeast*Pair* methods considering the produced values in Figure 3. When there are multiple pairs with the same accuracy, the most accurate pair was selected by considering the most divergent and the most sensitive pair in order.

Algorithm 2: Base learner selection algorithm.

```
Require:
    A set of all hypothesis: $\boldsymbol{\mathcal{H}} = \{\mathcal{H}_{sn}, \mathcal{H}_{sd}, \mathcal{H}_{dn}, \mathcal{H}_{dd}\}$
    Hypothesis per analysis types: $\mathcal{H}_i = \{H_{i_1}, H_{i_2}, H_{i_3}, H_{i_4}\}$ where $i = \{sn, sd, dn, dd\}$
Ensure:
    A subset of selected learners: $\boldsymbol{\mathcal{H}}' = \{\mathcal{H}'_{sn}, \mathcal{H}'_{sd}, \mathcal{H}'_{dn}, \mathcal{H}'_{dd}\}$
 1: $\boldsymbol{\mathcal{H}}' \leftarrow \{\}$
 2: for all $\mathcal{H} \in \boldsymbol{\mathcal{H}}$ do
 3:     $\mathcal{H}' \leftarrow \{\}$
 4:     $H_{acc} \leftarrow$ FindMostAccuratePair($\mathcal{H}$)
 5:     $H_{l_{acc}} \leftarrow$ FindLeastAccuratePair($\mathcal{H}$)
 6:     $H_{sns} \leftarrow$ FindMostSensitivePair($\mathcal{H}$)
 7:     $H_{l_{sns}} \leftarrow$ FindLeastSensitivePair($\mathcal{H}$)
 8:     Add $H_{acc}$, $H_{l_{acc}}$, $H_{sns}$ and $H_{l_{sns}}$ to $\mathcal{H}'$
 9:     Add $H'$ to $\boldsymbol{\mathcal{H}}'$
10: end for
```

Accuracy and sensitivity results are given in Table 1 for discussed three schemes. As seen, majority voting produced 100% sensitive output for both cases. However, its accuracy was not high as expected. Although selecting a base learner subset hasn't an obvious gain over using all base learners, it can be seen that the two most accurate (97.87%) and the two most sensitive (99.46%) outputs for stacking were obtained from the selected learners.

In Table 1, only accuracy and sensitivity values were given since the motivation of this study is to increase these metrics for malware detection problem. Other metrics can be inference intuitively from these values (e.g. specifity).

Table 1: Accuracy and sensitivity results of combination schemes

	Use All Base Learners		**Use Selected Base Learners**	
	Accuracy	**Sensitivity**	**Accuracy**	**Sensitivity**
***k*-NN**	97.33	98.91	97.07	98.91
NNet	97.33	98.91	**97.87**	99.46
SVMLinear	94.67	93.48	93.60	93.48
SVMPoly	96.80	98.91	97.07	**99.46**
SVMRadial	97.07	98.37	**97.87**	97.83
CART	96.00	97.28	96. 00	97.28
Majority Voting	91.47	100.0	90.40	100.0

Combining base learners' outputs for stacking requires a missing value handling operation because there will be extra samples in some base learners' outputs since different type of datasets were used. For instance, it isn't sensible to force a native code expert to produce a prediction about an application that doesn't have native code. Hence, we have applied a global constant replacement policy for such missing values to indicate that this expert doesn't have an opinion about related sample. "0" and "1" values were used to indicate *benign* and *malicious* predictions respectively, and "2" was used to indicate *no opinion*.

5.4 Evaluation

As seen in Figure 3, static native analysis gives very accurate and sensitive results. However, only less than half of the applications contain native code. Hence, main point for comparison must be Dalvik byte analysis. Static native Dalvik analysis' best accuracy was 97.22% and best sensitivity was 98.67% which are produced from different base learners. On the other hand, dynamic Dalvik analysis' best sensitivity was 100.0% but best accuracy was only 85.35%.

Ensemble system with the proposed selection algorithm was produced 97.87% accuracy and 99.46% sensitivity. This value is higher than static Dalvik, dynamic Dalvik and dynamic native analyses and also comparable with static native analysis. Furthermore, ensemble system provides more complete solution since different aspects was considered.

As seen from the evaluation results, ensemble learning provided more accurate and complete solution by considering different aspects of applications. On the other hand, since ensemble systems are constructed with lots of base learner, time needed to construct an ensemble system is larger than generating a single learner. However, once features are extracted and selected, base learners can be generated in parallel.

Feature extraction process for dynamic analyses generally takes much longer time than static analyses. In this study, static features were extracted within minutes but dynamic feature extraction took about a week using a single emulator. However, using more than one emulator or device can easily solve this problem.

6 CONCLUSION AND FUTURE WORK

The objective of this study was to solve malware detection problem by proposing architecture based on ensemble learning. For this purpose, different types of features were extracted with multiple methods and these features were processed by multiple mining algorithms in order to generate diverse base learners. Then, results of these base learners were combined in the scope of ensemble learning. Results show that using such architecture increases accuracy and sensitivity of detection operation.

Researchers have proposed lots of Android malware detection studies so far. However, a direct comparison of our results with these studies wasn't supplied for several reasons. First of all, one of the objectives of this study was to show benefits of ensemble learning for malware detection. Secondly, this study isn't a competitor of other detection tools; some tools can be adapted to this study as a feature extraction component so that they can be used to create new expert groups (e.g. [9]).

Since this study showed benefits of ensemble learning for malware detection problem, we will continue to add and try new components. For example, different evaluation criteria can be used in mining algorithms to improve accuracy and sensitivity. Or, different diversity measurements can be tried. We are also studying to improve accuracy by selecting a subset of benign applications among half a million applications.

7 ACKNOWLEDGMENT

We collected benign applications using "Google Play Crawler", which is developed by Ali Demiroz to support this study, thanks Ali. We also thanks to Zhou and Jiang for sharing their malware collection with us.

"apktool", "smali" and "baksmali" applications were used as reverse engineering tools. Additionally, "R" and its various packages were used for data mining algorithms.

REFERENCES

[1]. Alpaydin, E.: Introduction to Machine Learning (Adaptive Computation and Ma-chine Learning). The MIT Press (2004)

[2]. Blum, A.L., Langley, P.: Selection of relevant features and examples in machine learning. ARTIFICIAL INTELLIGENCE 97, 245-271 (1997)

[3]. Brown, G., Wyatt, J., Harris, R., Yao, X.: Diversity creation methods: A survey and categorisation. Journal of Information Fusion 6, 5-20 (2005)

[4]. Burguera, I., Zurutuza, U., Nadjm-Tehrani, S.: Crowdroid: Behavior-based mal-ware detection system for android. In: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices. pp. 15-26. SPSM '11, ACM, New York, NY, USA (2011)

[5]. Demiroz, A.: Google Play Crawler (2013), https://github.com/Akdeniz/ google-play-crawler, [Online; accessed 1-April-2014]

[6]. Dietterich, T.G., Bakiri, G.: Solving multiclass learning problems via error-correcting output codes. Journal of Artificial Intelligence Research 2, 263-286 (1995)

[7]. Dietterich, T.: Ensemble methods in machine learning. In: Multiple Classifier Sys-tems, Lecture Notes in Computer Science, vol. 1857, pp. 1{15. Springer Berlin Heidelberg (2000)

[8]. Deroski, S., enko, B.: Is combining classifiers with stacking better than selecting the best one? Machine Learning 54(3), 255-273 (2004)

[9]. Enck, W., Gilbert, P., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. In: Proceedings of the 9th USENIX conference on Operating systems design and implementation. pp. 1-6. OSDI'10, USENIX Association, Berkeley, CA, USA (2010

[10]. Enck, W., Ongtang, M., McDaniel, P.: On lightweight mobile phone application certification. In: Proceedings of the 16th ACM Conference on Computer and Com-munications Security. pp. 235-245. CCS '09, ACM, New York, NY, USA (2009)

[11]. Friedman, J., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting. Annals of Statistics 28, 2000 (1998)

[12]. Guyon, I.: An introduction to variable and feature selection. Journal of Machine Learning Research 3, 1157-1182 (2003)

[13]. Hansen, L., Salamon, P.: Neural network ensembles. Pattern Analysis and Machine Intelligence, IEEE Transactions on 12(10), 993-1001 (Oct 1990)

[14]. Ho, T.K.: The random subspace method for constructing decision forests. IEEE Trans. Pattern Anal. Mach. Intell. 20(8), 832-844 (Aug 1998)

[15]. Kantardzic, M.: Data Mining: Concepts, Models, Methods and Algorithms. John Wiley & Sons, Inc., New York, NY, USA (2002)

[16]. Kuncheva, L., Whitaker, C.: Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. Machine Learning 51(2), 181-207 (2003)

[17]. Kwok, S.W., Carter, C.: Multiple decision trees. In: Proceedings of the Fourth Annual Conference on Uncertainty in Artificial Intelligence. pp. 327-338. UAI '88, North-Holland Publishing Co., Amsterdam, The Netherlands, The Netherlands (1990)

[18]. Partridge, D., Yates, W.B.: Engineering multiversion neural-net systems. NEURAL COMPUTATION 8, 869-893 (1995)

[19]. Petrakos, M., Benediktsson, J.A., Kanellopoulos, I.: The effect of classifier agree-ment on the accuracy of the combined classifier in decision level fusion. IEEE T. Geoscience and Remote Sensing 39(11), 2539-2546 (2001)

[20]. Sahs, J., Khan, L.: A machine learning approach to android malware detection. In: Intelligence and Security Informatics Conference (EISIC), 2012 European. pp. 141-147 (Aug 2012)

[21]. dos Santos, E., Sabourin, R., Maupin, P.: Single and multi-objective genetic al-gorithms for the selection of ensemble of classifiers. In: Neural Networks, 2006. IJCNN '06. International Joint Conference on. pp. 3070-3077 (2006)

[22]. Schmidt, A.D., Schmidt, H.G., Clausen, J., Yksel, K.A., Kiraz, O., Camtepe, A., Albayrak, S.: Enhancing security of linux-based android devices. In: in Proceedings of 15th International Linux Kongress. Lehmann (Oct 2008)

[23]. Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., Weiss, Y.: andromaly: a behavioral malware detection framework for android devices. Journal of Intelligent Information Systems 38(1), 161-190 (2012)

[24]. Sharkey, A.J., Sharkey, N.E.: Combining diverse neural nets. THE KNOWLEDGE ENGINEERING REVIEW 12, 231-247 (1997)

[25]. Tang, E., Suganthan, P., Yao, X.: An analysis of diversity measures. Machine Learning 65(1), 247-271 (2006)

[26]. Teu, P., Kraxberger, S., Orthacker, C., Lackner, G., Gissing, M., Marsalek, A., Leibetseder, J., Prevenhueber, O.: Android market analysis with activation pat-terns. In: Prasad, R., Farkas, K., Schmidt, A., Lioy, A., Russello, G., Luccio, F. (eds.) Security and Privacy in Mobile Information and Communication Sys-tems, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 94, pp. 1-12. Springer Berlin Heidelberg (2012)

[27]. Tulyakov, S., Jaeger, S., Govindaraju, V., Doermann, D.: Review of classifier com-bination methods. In: In Machine Learning in Document Analysis and Recognition. Informatica 34 (2010) 111118 S. Vemulapalli et al (2008)

[28]. Wolpert, D.H.: Stacked generalization. Neural Networks 5, 241-259 (1992)

[29]. Yu, L., Liu, H.: Feature selection for high-dimensional data: A fast correlation-based filter solution. pp. 856-863 (2003)

[30]. Zhou, W., Zhou, Y., Jiang, X., Ning, P.: Detecting repackaged smartphone applica-tions in third-party android marketplaces. In: Proceedings of the second ACM con-ference on Data and Application Security and Privacy. pp. 317-326. ACM (2012)

[31]. Zhou, Y., Jiang, X.: Dissecting android malware: Characterization and evolution. In: Security and Privacy (SP), 2012 IEEE Symposium on. pp. 95-109 (May 2012)

[32]. Zhou, Y., Wang, Z., Zhou, W., Jiang, X.: Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. Proceedings of the 19th Annual Network and Distributed System Security Symposium pp. 5-8 (2012)

[33]. Zhou, Z.H., Wu, J., Tang, W.: Ensembling neural networks: Many could be better than all. Artificial Intelligence 137(12), 239-263 (2002)