International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:9, No:5, 2015

A Comparative Study of Malware Detection Techniques Using Machine Learning Methods

Cristina Vatamanu, Doina Cosovan, Dragoş Gavriluţ, Henri Luchian

Abstract**—**In the past few years, the amount of malicious software increased exponentially and, therefore, machine learning algorithms became instrumental in identifying clean and malware files through (semi)-automated classification. When working with very large datasets, the major challenge is to reach both a very high malware detection rate and a very low false positive rate. Another challenge is to minimize the time needed for the machine learning algorithm to do so. This paper presents a comparative study between different machine learning techniques such as linear classifiers, ensembles, decision trees or various hybrids thereof. The training dataset consists of approximately 2 million clean files and 200.000 infected files, which is a realistic quantitative mixture. The paper investigates the above mentioned methods with respect to both their performance (detection rate and false positive rate) and their practicability.

Keywords**—**Detection Rate, False Positives, Perceptron, One Side Class, Ensembles, Decision Tree, Hybrid methods, Feature Selection.

I. Introduction

A HUGE amount of malware exists for every platform and operating system. Once the malware is identified, antivirus software can automatically detect and clean the malware. Identifying previously unknown malware also needs to be done in an automatic manner, due to the enormous amount of new malware (of the order of magnitude of 10^5) that is launched daily. This is why machine learning took the proscenium in malware detection. This paper is concerned with several critical issues in detecting new malware.

The first issue is the rate of false positives. The wrong classification of a legitimate file (labeled as malicious) can lead to crashes, data loss or other consequences which can be comparable to or even worse than the effect of a wrong classification of a malicious file (labeled as legitimate). Therefore, a main practical concerns of any machine learning approach to malware detection is to keep the rate of false positives very low.

Obviously, the issue of the detection rate remains critical: the higher the rate of files correctly labeled as malware, the better the user is protected against cyber-attacks. Nevertheless, many classification algorithms reach a high detection rate at the expense of the false positives rate, which increases as well. The major challenge is to find a trade-off between detection rate and false positive rate. This trade-off is all the more difficult to achieve since the available training time is severely limited by the very fast pace of malware evolution: new versions of the same malware become active every hour. Databases of millions of records, each of them having thousands of features, tighten further the time constraints.

This paper is based on our previous study ([3]), that presents the One Side Class (OSC) algorithm. It was optimized in terms of training time; in this paper we consider its third version, called OSC-3. Its main downside is the average detection rate, so it can't be used as a standalone detection method. In the sequel, several approaches (feature selection, ensemble strategies, binary decision trees, hybridization) are discussed, with an eye towards reaching a good trade-off between training time, detection rate and false positive rate.

The perceptron algorithm was chosen as base algorithm because it fits well when integrating in an AV product (low time complexity, the ability to run in a parallel manner), but it has to run with the given features (it doesn't create new features as, for example, genetic algorithms). In this case, a good data representation is crucial in obtaining the best results. An important step of our research was creating the database, choosing the right features for describing our data and refining our database by removing the unwanted noise.

The experiments show an improvement of the detection rate from 29.78% to 76.46% while maintaining a medium rate of 0.5080% false positives in 416 minutes.

In Section II, we review solutions proposed in the literature for some of the problems we are dealing with, while in Section III we present the dataset and the algorithms (the base algorithm OSC, ensembles and hybrid algorithms based on OSC). In the last section, we discuss the experimental results.

II. Related Work

This section reviews approaches and techniques proposed by various authors regarding central issues for our experiments: false positive rate minimization and ensemble techniques.

A. False Positive Rate Minimization

A few approaches based on cost-sensitive classification have been proposed for reducing false positives. Content-specific misclassification costs have been used in association with Support Vector Machines (SVMs) in [6], but have also been generally described in [2], where the machine learning

Cristina Vatamanu, Doina Cosovan and Dragos Gavrilut are with the Faculty of Computer Science from "Al. Ioan Cuza" University, Iasi, Roumania and with the Bitdefender Anti-Malware Laboratory (phone: +40-724-334-632 ; fax:+40-232-244-390; e-mail: cvatamanu@bitdefender.com)

Henri Luchian is with the Faculty of Computer Science from "Al. Ioan Cuza" University, Iasi, Roumania

algorithm is considered a black box and its implementation details are left out.

Another interesting mechanism for achieving a low false positive rate is presented in [9], where the results of 53 different spam filters are combined instead of using only one. The mechanisms used include averaging the binary classifications returned by the individual filters, averaging log-odds estimates based on the scores returned by the individual filters and logistic-regression-based stacking methods.

In [17], two techniques aiming at limiting both false positive and false negative rates for spam detection are presented. One of them consists in stratification, which involves weighing good messages as more valuable than spam data. The other one, firstly discards the samples that can be easily classified and then lets the classifier focus on the ones which are similar but pertain to different classes. This way, the second model focuses on the region which is harder to classify. Both techniques were tried on Naive Bayes and logistic regression. In this paper, we discuss an a pproach similar to the second one, using perceptrons.

In [16], different machine learning algorithms have been tested against spam detection. Focusing on limiting the number of false positives, the authors concluded that perceptrons can't be altered in such a way as to limit the number of falsely detected emails.

Despite this conclusion, in [4], a zero false positive perceptron was proposed and studied. It uses as basis a standard perceptron algorithm, but adds a few steps to be executed after each iteration and aiming at adjusting the separation plan in such a way as to correctly classify all the elements from one of the two classes. In the end, the plan will separate on one part only the malicious files and on the other - both clean and malicious files. This perceptron was further optimized and tested in [3]. The authors in [1] analyze the possibility that the zero false positive constraint can cause over-fitting. According to their conclusion, the detection rate decreases normally after a period of time, so the generated model can be used to classify malware even after a period of time passes.

B. Ensemble Techniques

Lately, a lot of machine learning and data mining techniques have been proposed in order to improve proactivity of malware detection ([14], [15], [7], [11], [5], [12]). Some of them performed better than the others on different aspects of interest. In the literature appeared the idea that it would be useful to combine the advantages provided by those techniques into a single classifier. Thus different ensembles were proposed to deal with malicious files, network traffic and even malicious urls.

In [18], the Dempster-Shafer theory is used to create combining rules for individual decisions of PNN (Probabilistic Neural Network) classifiers. 450 malicious executables from the VX Heavens (http://www.vx.netlux.org) malware dataset and 423 benign files obtained from a Windows 2000 server machine were used to compare individual PNN classifiers with the obtained ensemble on n-gram based features selected using the Information Gain method. According to their results, the ensemble performed better.

A new ensemble learning method, called SVM-AR, is proposed in [8]. It combines SVM (Support Vector Machines) and association rules based on hierarchical taxonomy. First, SVM determines a hyper plane that classifies the samples in two classes: clean and malicious. Then, the association rules are used as local optima filters in order to solve false predictions made by the SVM algorithm. According to the authors, this ensemble is comparable to any individual learning algorithm when it comes to execution times and even performs better than the well-known ensemble algorithms (Bagging, Boosting, Voting, Stacking, Grading) in respect to accuracy, detection and false positive rates.

In [10], a multi-inducer ensemble is proposed. Starting with the idea that combining the output of different classifiers is only useful when they disagree, the authors decided to combine five different classifiers corresponding to five different classifier families: classifier C4.5 corresponding to classifier family Decision Trees, KNN - Lazy Classifiers, VFI - General Classifiers, OneR - Rules, Naive Bayes - Bayes Classifier. These are expected to generate models that are going to classify differently given the same input. Different combination techniques were also used: best classifier (choosing the classifier that outperforms all the others), Majority Voting, Performance Weighting, Distribution Summation, Bayesian Combination, Naive Bayes, Stacking and Troika. In the end, they were tested on 22.735 benign and 7.690 malicious files in terms of accuracy, UAC and detection time. PE (Portable Executable) and function-based features and n-grams were used after a selection based on Document Frequency, Fisher Score and Gain Ratio. Bayesian Combination managed to combine well all the three requirements, although it was surpassed by different combination methods regarding one of the three tested aspects.

Ensembles have been also used for detection of Android malware. An example consists [13], where combination schemes like Majority Voting, Stacking with all the base learners and Stacking with a subset of base learners selected using a simple heuristic are used to aggregate base learners generated using algorithms like k-NN, NNet, SVMLinear, SVMPoly, SVMRadical, and CART. The tests were performed on 1225 malicious and 1225 clean samples downloaded from Google Play, using as features static and dynamic native API calls and static and dynamic Dalvik byte API calls. The authors concluded that the selection of a subset of base learners doesn't bring a significant improvement to the results. However, we are more interested in reducing the false positive rate, while the above mentioned authors were mainly interested in improving the detection rate. Their view was that a false negative would cause a malware to act freely, while a false positive would be easily solved by the security experts of the application market. However, in the frame of our work, a false positive can leave a computer unusable which takes precedence over any other concern.

III. Database and Algorithms

A. Database

The study was conducted on a database with millions of records, 2.348.707 to be precise. It is divided as follows: 242.811 represent the malicious files and 2.105.896 records correspond to the clean files. The malware collection was gathered in the last three months, representing fresh new samples from various malware families. The clean files were collected for a period of one year and represent different legitimate applications, system files from many operating systems, components of the mostly used software. Each record has 6.275 boolean features, that were extracted both statically and dynamically. Static features include different information related to file geometry, type of packer, type of compiler, executable flags, and so on.

The dynamic features are extracted after the files are executed or loaded. Examples of features extracted in this manner could be: if the file drops other files on the disk, if it connects to internet, if it downloads other components, if it injects itself in some system processes.

In order to obtain this database, a filtering process was applied for the malware collection. Certain types of malware were removed from the dataset: file infectors, adware, keygens. The behavior of file infectors can be briefly described as modifying clean files by adding malicious code. Given this fact the malware files are very similar to the original clean files, so they represent a noise in the database. This is also the case of adware: they appear to be legitimate programs or applications, but in certain conditions unwanted commercial content is displayed.

B. OSC

The OSC algorithm is a modified perceptron, which has the property of correctly classifying all the records from one class. At the end of each training iteration, there is an additional step, that modifies the hyper-plan in such a way as to ensure that all the elements pertaining to a certain class are all correctly classified.

We present below a combination of two of these algorithms: one for the class of clean files and one for the class of malicious files. As a result, at the end of each iteration of the training process, two hyper-planes will be defined: one will correctly classify all the records corresponding to the clean class and the other - all the records from the malware class. As an example, after applying the two OSC algorithms, the two hyper-plans will look like in the Fig. 1.

The hyper-plan for the clean class (blue plan) will have above it only malicious records correctly classified and below it - all the clean records and a few malicious records which were incorrectly classified. On the other hand the hyper-plan for the malware class (red plan) will have below clean samples correctly classified and above - all the malicious samples and some of the clean samples which were incorrectly classified. By the end of the training process, all the elements above the blue plan and all the elements below the red plan are correctly classified. Therefore this two level OSC algorithm can be integrated in an ensemble-based system. The correct classified records will be removed from the data set at each ensemble iteration. Only the records between the two plans will continue to participate at the next ensemble iteration. In theory, if the number of iterations is high enough, all the records will be correctly classified (Fig. 2).

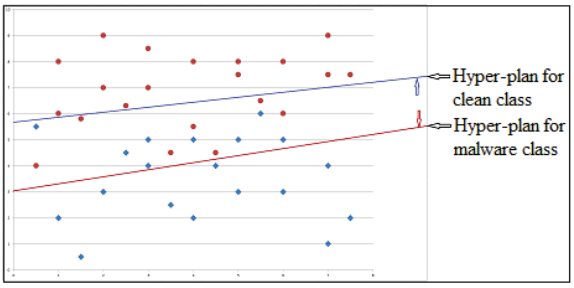


Fig. 1. The two hyper-plans at the end of the training process

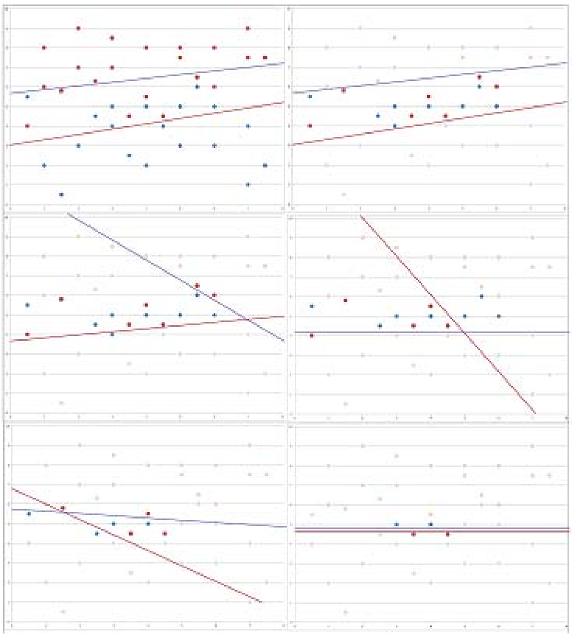


Fig. 2. Complete ensemble process

C. Ensembles

Ensemble Learning involves combining various learning algorithms so that the final results are better compared to using each of them standalone. The term ensemble, however, is used to refer methods that use the same learning algorithm in order to generate multiple hypotheses. The idea behind ensembles is that different features can describe well different areas of the dataset. This means that one can build different models that rely on different features in order to describe different areas of the dataset. The Algorithm 1 illustrates our approach.

In this paper, we build ensembles based on the OSC perceptron. At each iteration, an OSC perceptron is trained and the dataset needed for the next iteration is extracted. The algorithm stops either when all the samples from the training set are correctly classified or when the maximum number of iterations is reached. In order to choose the best way of

Algorithm 1 Ensemble Algorithm

```
$D \leftarrow \bigcup_{i=1}^{|R|} R_i$ - the database
$A \leftarrow \{A_1, A_2, ...A_n\}$ - set of algorithms
function $ensemble(D, A)$
    while $|D| > 0$ do
        Choose $A_i$ from $A$, taking into consideration $D$
        $S \leftarrow A_i(D), 0 < |S| < |D|$
        $D \leftarrow D \setminus S$
    end while
end function
```

Algorithm 2 Ensemble One Class Algorithm

```
$S \leftarrow \bigcup_{i=1}^{|R|} R_i$ - the database
DL - desired class label for the OSC algorithm
maxCount - the maximum number of iterations
function $ensemble\_one\_class(S, DL, maxCount)$
    $itCount \leftarrow 0$
    repeat
        $NS \leftarrow \emptyset$
        $Model \leftarrow OSC3(S, DL)$
        for $i \leftarrow 1 \rightarrow |S|$ do
            if $IsCorrectlyClassified(Model, S_i)$ then
                if $S_i.L = DL$ then
                    $NS.push(S_i)$
                end if
            else
                $NS.push(S_i)$
            end if
        end for
        $S \leftarrow NS$
        $itCount \leftarrow itCount + 1$
    until $|S| = 0 \parallel itCount = maxCount$
end function
```

composing the OSC perceptrons into an ensemble, we tried a few different techniques of changing the datasets across iterations. In order to simplify the writing of the algorithms corresponding to these techniques, the following notations are used:

1) $Record \rightarrow R$
2) $Features \rightarrow F$
3) $Label \rightarrow L$

The simplest mechanism ignores the data that was correctly classified as being part of one of the two classes (clean or infected) during the current iteration, so that, in the next iteration, the dataset contains only the misclassified samples and the samples correctly classified as pertaining to the other class.

For example, the Algorithm 2 trains a model on the current dataset and, at each iteration, prepares the dataset for the next iteration by discarding the samples correctly classified from the class opposed to desired class DL (Fig. 3). Of course, DL can take one of two possible values: clean and malware.

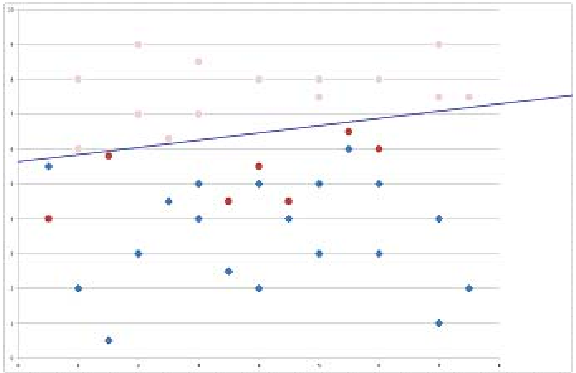


Fig. 3. First iteration for Ensemble One Class Algorithm

Algorithm 3 Ensemble Alternative Class Algorithm

```
$S \leftarrow \bigcup_{i=1}^{|R|} R_i$ - the database
DL - desired class label for the OSC algorithm
maxCount - the maximum number of iterations
function $ensemble\_alternative\_class(S, DL, maxCount)$
    $itCount \leftarrow 0$
    repeat
        $NS \leftarrow \emptyset$
        $Model \leftarrow OSC3(S, DL)$
        for $i \leftarrow 1 \rightarrow |S|$ do
            if $IsCorrectlyClassified(Model, S_i)$ then
                if $S_i.L = DL$ then
                    $NS.push(S_i)$
                end if
            else
                $NS.push(S_i)$
            end if
        end for
        $S \leftarrow NS$
        $DL \leftarrow (-1) * DL$
        $itCount \leftarrow itCount + 1$
    until $|S| = 0 \parallel itCcount = maxCount$
end function
```

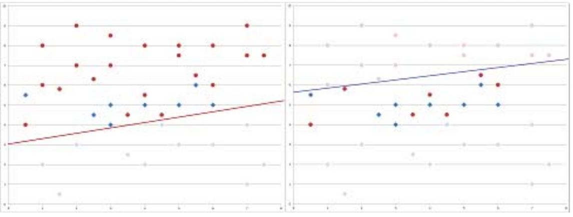


Fig. 4. First two iterations for Ensemble Alternative Class Algorithm

The previously described techniques tend to train, at each step, models on all the samples of a given class and the remaining ones from the other class. This happens because all the files of a given type are kept until the end and the files of the other type are discarded systematically. This type of ensembles can be viewed as filters for clean and respectively malicious files. In the last iterations, the algorithm will have to build a filter for a small set of files of one type and a very big amount of files of the other type. But, sometimes it is hard to find properties of the small set so that none of the files from the huge set has in order to discard the small set. Therefore we combined these two algorithms in a single one.

Algorithm 4 Ensemble n Alternative Class Algorithm

```
$S \leftarrow \bigcup_{i=1}^{|R|} R_i$ - the database
DL - desired class label for the OSC algorithm
maxCount - the maximum number of iterations
n - number of consecutive iterations
function $ensemble\_n\_alternative\_class(S, DL, maxCount, n)$
    $count \leftarrow 0$
    $itCount \leftarrow 0$
    repeat
        $NS \leftarrow \emptyset$
        $Model \leftarrow OSC3(S, DL)$
        for $i \leftarrow 1 \rightarrow |S|$ do
            if $IsCorrectlyClassified(Model, S_i)$ then
                if $S_i.L = DL$ then
                    $NS.push(S_i)$
                end if
            else
                $NS.push(S_i)$
            end if
        end for
        $S \leftarrow NS$
        $count \leftarrow count + 1$
        if $count = n$ then
            $DL \leftarrow (-1) * DL$
            $count \leftarrow 0$
        end if
        $itCount \leftarrow itCount + 1$
    until $|S| = 0 \parallel itCcount = maxCount$
end function
```

Algorithm 5 Ensemble Best Algorithm

```
$S \leftarrow \bigcup_{i=1}^{|R|} R_i$ - the database
DL - desired class label for the OSC algorithm
maxCount - the maximum number of iterations
function $ensemble\_best(S, DL, maxCount)$
    $itCount \leftarrow 0$
    repeat
        $NS_1 \leftarrow \emptyset$
        $NS_2 \leftarrow \emptyset$
        $Model_1 \leftarrow OSC3(S, DL)$
        $Model_2 \leftarrow OSC3(S, (-1) * DL)$
        for $i \leftarrow 1 \rightarrow |S|$ do
            if $IsCorrectlyClassified(Model_1, S_i)$ then
                if $S_i.L = DL$ then
                    $NS_1.push(S_i)$
                end if
            else
                $NS_1.push(S_i)$
            end if
            if $IsCorrectlyClassified(Model_2, S_i)$ then
                if $S_i.L = DL$ then
                    $NS_2.push(S_i)$
                end if
            else
                $NS_2.push(S_i)$
            end if
        end for
        if $|NS_1| > |NS_2|$ then
            $S \leftarrow NS_1$
        else
            $S \leftarrow NS_2$
        end if
        $DL \leftarrow (-1) * DL$
        $itCount \leftarrow itCount + 1$
    until $|S| = 0 \parallel itCcount = maxCount$
end function
```

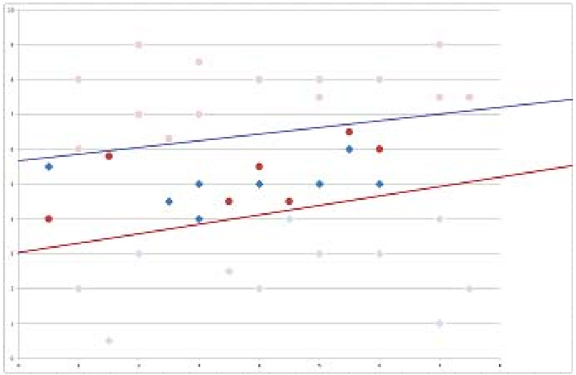


Fig. 5. First iteration for Ensemble Both Classes Algorithm

We consider two ways of doing this. The first one involves discarding alternatively clean and malicious files in successive iterations as illustrated in Algorithm 3. At each iteration, one has to make sure that the class of interest in changed, as illustrated in Fig. 4. In this case, one class is considered to be positive (1) and the other one - negative (-1).

The second one switches every k iterations between discarding clean files and discarding infected files, where $k > 1$. By modifying the previous algorithm, we obtained the Algorithm 4, that illustrates this technique.

In order to understand the limitations imposed by these strategies, let us consider the following use case. During a specified iteration, in which the algorithm is supposed to discard the malicious files, the resulted model classifies correctly a lot of clean files and only a few malicious files. In this way, the opportunity to discard a big amount of files is lost just because the choice of the class of files to be discarded is independent of the results obtained by the current model. In order to solve this problem, the Algorithm 5 is proposed. It decides to discard the class of files that has the biggest number of correct classifications in that specific iteration.

Since any model obtained during an iteration classifies correctly both clean and infected files, it would be reasonable to try a strategy where all the correctly classified files are discarded. The advantage consists in the fact that the samples that are already correctly classified do not alter the models to be built in the next iterations. Thus the other models can focus only on the data wrongly classified during the previous iteration. Also the amount of files from the dataset decreases faster. This approach is presented in Algorithm 6 (Fig. 5).

An interesting aspect to be considered here is the feature

Algorithm 6 Ensemble Both Classes Algorithm

```
$S \leftarrow \bigcup_{i=1}^{|R|} R_i$ - the database
DL - desired class label for the OSC algorithm
maxCount - the maximum number of iterations
function $ensemble\_both\_classes(S, DL, maxCount)$
    $itCount \leftarrow 0$
    repeat
        $NS \leftarrow \emptyset$
        $Model_1 \leftarrow OSC3(S, DL)$
        $Model_2 \leftarrow OSC3(S, (-1) * DL)$
        for $i \leftarrow 1 \rightarrow |S|$ do
            if $IsCorrectlyClassified(Model_1, S_i)$ then
                if $notIsCorrectlyClassified(Model_2, S_i)$ then
                    $NS.push(S_i)$
                end if
            else
                if $IsCorrectlyClassified(Model_2, S_i)$ then
                    $NS.push(S_i)$
                end if
            end if
        end for
        $S \leftarrow NS$
        $itCount \leftarrow itCount + 1$
    until $|S| = 0 \parallel itCcount = maxCount$
end function
```

set. Having a large amount of features describing the data to be classified imposes the necessity of feature selection. Normally, feature selection is performed only once, right before training a classifier. However, when it comes to ensembles, one can select a different set of features after each iteration. In this way, every model can be built on the features that are more appropriate for the subset it is trained on.

Finally, we tried a hybrid approach. We tested different scenarios where we combined a Binary Decision Tree with several ensemble techniques based on the OSC algorithm.

D. Binary Decision Trees

A Binary Decision Trees consists of a root node, several decision nodes and terminal nodes (leafs). At each decision node condition on a certain feature is evaluated and the data set is consequently divided into two subsets. The terminal nodes represent classes of records with specific properties (clusters). The set of features used in successive evaluations are chosen using various rules / scores. Subsequently, each set of records from the terminal nodes of the binary tree (the clusters) are used in the training process for different classification algorithms or systems. Algorithm 7 describes the method used to create these subsets.

The save operation represents the fact that all the elements from records are comprised in a new tree node (a terminal node).

An important issue when using a Binary Decision Tree is the choice of the score functions. Since we wanted to achive a equal repatization of records on all terminal nodes we've used the following score (Median Close):

Algorithm 7 Binary Decission Tree Split

```
$S \leftarrow \bigcup_{i=1}^{|R|} R_i$ - the database
$F \leftarrow \bigcup_{j=1}^{|R.F|} R.F_j$ - the features set
maxDepth - the maximum binary tree leves
$depth \leftarrow 0$
function $cluster$(S, F, depth)
    if $depth = maxDepth$ then
        $save(cluster,\ S)$
        $return$
    end if
    $max \leftarrow 0$
    $EF \leftarrow 0\ (evaluated\ feature)$
    for $j \leftarrow 1 \rightarrow |F|$ do
        $score \leftarrow compute\_score(S,\ F_j)$
        if $score > max$ then
            $max \leftarrow score$
            $EF \leftarrow F_j$
        end if
    end for
    $S_R \leftarrow \emptyset$
    $S_L \leftarrow \emptyset$
    for $i \leftarrow 1 \rightarrow |S|$ do
        if $S_i.EF = 0$ then
            $S_L \leftarrow S_L \bigcup S_i$
        else
            $S_R \leftarrow S_R \bigcup S_i$
        end if
    end for
    $F \leftarrow F \setminus EF$
    $cluster(S_L,\ depth\ +\ 1,\ F)$
    $cluster(S_R,\ depth\ +\ 1,\ F)$
end function
```

$$\frac{(1-|\frac{countClean}{totalClean}-0.5|)+(1-|\frac{countInfected}{totalInfected}-0.5|)}{2}$$

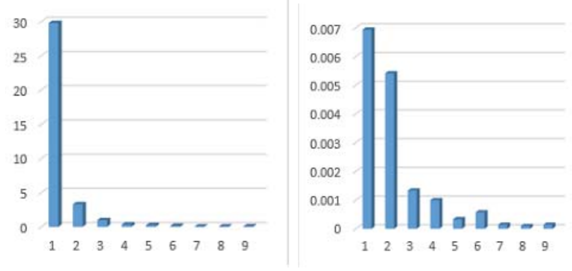
The following notation was used:

- countClean - the number of clean files for which the given feature is set to true
- totalClean - the total number of clean files
- countInfected - the number of infected files for which the given feature is set to true
- totalInfected - the total number of infected files

The preprocessing of the database was performed in order to obtain a set of clusters. The next step was to look for the optimal balance between training time, detection rate and false positive rate, using several classification approaches. The binary decision tree was combined with the two layer OSC algorithm and the ensemble systems discussed above; the Result section highlights the best solutions.

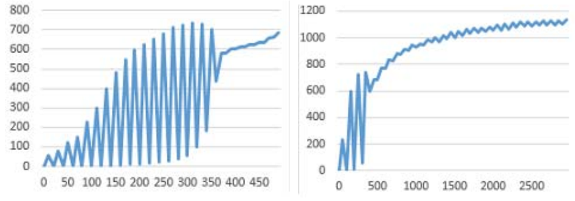
IV. Results

In order for the results to be consistent, all the tests were conducted on the same computer. As mentioned in the previous



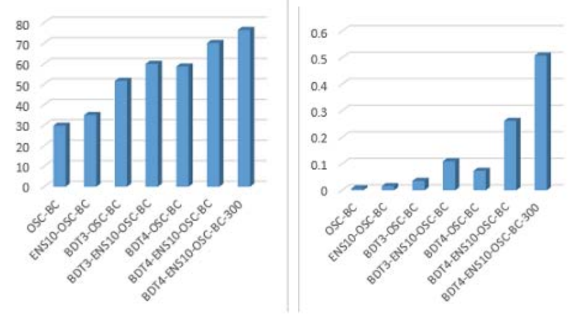
(a) (b)

Fig. 6. Detection Rate (a) / FP Rate (b) per iteration for ENS10-OSC-BC



(a) (b)

Fig. 7. Training process for OSC-BC with 500 iterations (a) and 3000 iterations (b) during the second step of BDT4-ENS10-OSC-BC



(a) (b)

Fig. 8. Detection Rate (a) and False Positive Rate (b) for the tested algorithms

section, the database consists of 2.348.707 records (242.811 malware, 2.105.896 clean). Each record has 6.275 features. The 500 features used in the training process were selected using F2 score. Each algorithm was tested using a 3-fold cross validation.

For brevity, we use the following abreviations:

1) OSC-BC → OSC Both Class
2) ENS10-OSC-BC → OSC Both Class in an ensemble system with 10 iterations (Alg. 6)
3) BDT3-OSC-BC → Binary Decision Tree (3 levels) combined with OSC-BC
4) BDT3-ENS10-OSC-BC → Binary Decision Tree (3 levels) combined with ENS10-OSC-BC
5) BDT4-OSC-BC → Binary Decision Tree (4 levels) combined with OSC-BC
6) BDT4-ENS10-OSC-BC → Binary Decision Tree (4 levels) combined with ENS10-OSC-BC
7) BDT4-ENS10-OSC-BC-3000 → Binary Decision Tree (4 levels) combined with ENS10-OSC-BC, but the OSC-BC is trained during 3000 iterations

Of the ensemble techniques presented in this paper, the Algorithm 6 achieved the best results. However, the improvement is only significant in the first 2-3 iterations, as illustrated by Fig. 6.

TABLE I
RESULTS

Algorithm	Detection Rate (percentage)	False Positives (percentage)	Training Time (in minutes)
OSC-BC	29.78 %	0.0069 %	125
ENS10-OSC-BC	34.96 %	0.0159 %	189
BDT3-OSC-BC	51.65 %	0.0355 %	200
BDT3-ENS10-OSC-BC	59.94 %	0.1093 %	256
BDT4-OSC-BC	58.66 %	0.0734 %	112
BDT4-ENS10-OSC-BC	70.07 %	0.2618 %	288
BDT4-ENS10-OSC-BC-3000	76.46 %	0.5080 %	416

Since ENS10-OSC-BC proved to be the best ensemble, the hybrid algorithms combine this ensemble with the Binary Decision Tree algorithm. For comparison purposes, a hybrid algorithm combining the OSC-BC algorithm with the Binary Decision Tree is tested as well.

Although a detection boost is obvious when combining the OSC-BC with the Binary Decision Tree, this may not be good enough. Consequently, we decided to analyze the data from each cluster. We noticed that, during the OSC training process, after the first ensemble iteration, approximately 70% of the iterations have an odd behavior: the number of correctly classified records alternated between a small value and a high value. Only during the last 30% of the iterations, the detection rate seemed to have a linear increase.This observation encouraged us to increase the number of iterations of the OSC-BC training process from 500 to 3000 starting from the second ensemble step.In this way, only 10% of the iterations presented the previously described behavior (detection rate alternations) and the detection boost was more visible. Specifically, detection rate of the second ensemble step increased from 16.99% to 27.84%. Fig. 7 illustrates this improvement.

The Fig. 8 illustrates both the detection and false positive rates for all the relevant algorithms, while the Table I summarizes the results values.

V. Conclusions

The algorithms presented above illustrate different ensemble configurations in order to boost the detection rate. Our experiments show that, while BDT4-ENS10-OSC-BC-3000 provides the best detection rate, it also has the highest false positive rate. Therefore, this algorithm can be used in practice but only in combination with a method for false positive filtering (such as file white-listing).

Most AV products have a detection based on multiple algorithms. There are two main reasons for this: one is to increase the detection rate and the other one is to protect against targeted malware. A targeted malware is a malware that is constructed in such a way that, at the moment it is deployed, it is not detected by a specific AV product. From the AV vendor point of view, the solution is not to have a detection based on complementary algorithms, but rather to have a detection where the same malware is detected by a completely different algorithm. This will make the task of a targeted malware more difficult. However, it poses a problem for the vendor as well. In order for a detection algorithm to be used in such a combination, it by itself should have a good detection rate (more than 50% of the samples). The method presented in this paper illustrates a way to increase the detection rate to a point where machine learning detection can be used to prevent AV malware targeted attacks.

Another thing that should be pointed out here is that using a Binary Decision Tree means that every subset that is generated by a terminal node can be tested in parallel. Even if this is merely a conclusion based on the design of a Binary Decision Tree, it is definitely something that should be considered for use in practice (for example, BDT4-ENS10-OSC-BC-3000 could be trained 16 times faster).

As future work, we consider developing a hybrid algorithm that combines Binary Decision Trees with ensemble algorithms in such a way as to perform binary splits only when the ensemble performs poorly. Also it would be interesting to use genetic algorithms for choosing at each ensemble step the algorithm to be used for the next ensemble step.

References

[1] Mihai Cimpoesu, Dragos Gavrilut, and Adrian Popescu. The proactivity of perceptron derived algorithms in malware detection. *Journal in Computer Virology*, 8(4):133–140, 2012.

[2] Pedro Domingos. Metacost: A general method for making classifiers cost-sensitive. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 15-18, 1999*, pages 155–164, 1999.

[3] Dragos Gavrilut, Razvan Benchea, and Cristina Vatamanu. Optimized zero false positives perceptron training for malware detection. In *14th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, SYNASC 2012, Timisoara, Romania, September 26-29, 2012*, pages 247–253, 2012.

[4] Dragos Gavrilut, Mihai Cimpoesu, Dan Anton, and Liviu Ciortuz. Malware detection using machine learning. In *Proceedings of the International Multiconference on Computer Science and Information Technology, IMCSIT 2009, Mragowo, Poland, 12-14 October 2009*, pages 735–741, 2009.

[5] Yongtao Hu, Liang Chen, Ming Xu, Ning Zheng, and Yanhua Guo. Unknown malicious executables detection based on run-time behavior. In *Fifth International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2008, 18-20 October 2008, Jinan, Shandong, China, Proceedings, Volume 4*, pages 391–395, 2008.

[6] Aleksander Kocz and Joshua Alspector. Svm-based filtering of e-mail spam with content-specific misclassification costs. In *IN PROCEEDINGS OF THE WORKSHOP ON TEXT MINING (TEXTDM2001*, 2001.

[7] Jeremy Z. Kolter and Marcus A. Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 6:2721–2744, 2006.

[8] Yi-Bin Lu, Shu-Chang Din, Chao-Fu Zheng, and Bai-Jian Gao. Using multi-feature and classifier ensembles to improve malware detection. *Journal of C.C.I.T.*, 39(2), 2010.

[9] Thomas R. Lynam, Gordon V. Cormack, and David R. Cheriton. On-line spam filter fusion. In *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, USA, August 6-11, 2006*, pages 123–130, 2006.

[10] Eitan Menahem, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Improving malware detection by applying multi-inducer ensemble. *Computational Statistics & Data Analysis*, 53(4):1483–1494, 2009.

[11] Robert Moskovitch, Yuval Elovici, and Lior Rokach. Detection of unknown computer worms based on behavioral classification of the host. *Computational Statistics & Data Analysis*, 52(9):4544–4566, 2008.

[12] Robert Moskovitch, Clint Feher, Nir Tzachar, Eugene Berger, Marina Gitelman, Shlomi Dolev, and Yuval Elovici. Unknown malcode detection using OPCODE representation. In *Intelligence and Security Informatics, First European Conference, EuroISI 2008, Esbjerg, Denmark, December 3-5, 2008. Proceedings*, pages 204–215, 2008.

[13] Mehmet Ozdemir and Ibrahim Sogukpinar. An android malware detection architecture based on ensemble learning. *Transactions on Machine Learning and Artificial Intelligence*, 2(3), 2014.

[14] Matthew G. Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J. Stolfo. Data mining methods for detection of new malicious executables. In *2001 IEEE Symposium on Security and Privacy, Oakland, California, USA May 14-16, 2001*, pages 38–49, 2001.

[15] Dong-Her Shih, Hsiu-Sen Chiang, and David C. Yen. Classification methods in the detection of new malicious emails. *Inf. Sci.*, 172(1-2):241–261, 2005.

[16] Konstantin Tretyakov. Machine learning techniques in spam filtering. *Data Mining Problem-oriented Seminar*, 3(177):60–79, 2004.

[17] Wen-tau Yih, Joshua Goodman, and Geoff Hulten. Learning at low false positive rates. In *CEAS 2006 - The Third Conference on Email and Anti-Spam, July 27-28, 2006, Mountain View, California, USA*, 2006.

[18] Boyun Zhang, Jianping Yin, Jingbo Hao, Dingxing Zhang, and Shulin Wang. Malicious codes detection based on ensemble learning. In *Autonomic and Trusted Computing, 4th International Conference, ATC 2007, Hong Kong, China, July 11-13, 2007, Proceedings*, pages 468–477, 2007.