Hybrid AdaBoost Classifiers for the Detection of PHP Webshells

Alexandre Croix
Research Unit Cyber Defense
Royal Military Academy
Brussels, Belgium
a.croix@cylab.be

Thibault Debatty
Research Unit Cyber Defense
Royal Military Academy
Brussels, Belgium
t.debatty@cylab.be

Wim Mees
Research Unit Cyber Defense
Royal Military Academy
Brussels, Belgium
w.mees@cylab.be

Louis Detry
UMons
Mons, Belgium
louis.detry@gmail.com

***Abstract*—In this work, we trained, measure and compare the efficiency of several classifiers: k-Nearest Neighbours, Multi-Layer Perceptron, Support Vector Machines and Decision Trees. We perform a complete parametric study on these classifiers to determine their optimal parameters in order to optimize their performance. The classifiers were trained and tested on a dataset made of PHP files previously analyzed by a multi-agents PHP webshells detector developed at the Royal Military Academy. Based on these well-known classifiers, we built several versions of AdaBoost algorithms to compare its performance with single optimized classifiers on different criteria: Area under the Curve of a Receiver Operating Characteristics curve, the Area Under the Curve of a Precision-Recall curve and F-Measure. We globally obtained good results, particularly with the Multi-Layer Perceptron. Despite good results, the AdaBoost approach underperforms some of the single-optimized classifiers.**

***Index Terms*—Decision Tree, KNN, Webshells, MLP, SVM, AdaBoost**

I. Introduction

Cyber-security has become a critical issue in today's digital age, with a growing number of cyber-attacks reported daily. From identity theft to data breaches, the threat landscape is constantly evolving, and organizations must be proactive in defending against these attacks. One crucial aspect of cyber-security is identifying potential threats and taking appropriate action to prevent them. This is where machine learning algorithms can play a crucial role. These algorithms can classify an event as a threat or not. They can use them to help a human analyst by presenting the elements having the highest probability of being a risk.

Using machine learning in cyber-security is sometimes tricky because of the lack of reliable data. Indeed, it is difficult to obtain a large amount of actual malware, APT, and webshells;... Generating artificial data or creating new instances based on existing data is a challenge compared to other data types, such as images.

In this paper, with present and perform a parametric study on several machine learning algorithms to compare their performance. These algorithms are Decision Tree (DT), k-Nearest Neighbor (KNN), Multi-Layer Perceptron (MLP) and Support Vector Machine (SVM). Then, we built a hybrid Adaptative Boosting algorithm that combines the four previous classifiers in several configurations.

These classifiers were evaluated on a specific task: classifying PHP files as webshells or harmless files. Before the classification itself, all the PHP files go through a Webshell detector. This detector is composed of 5 specific agents that produce a score between 0 and 1. These values are used as input for the different classifiers.

The rest of the paper is arranged as follows: Section (II) will give a basic description and explain the working of these classifiers. Then, Section (III) presents the structure of our hybrid Adaptative Boosting algorithm. In Section (IV), we evaluate the performance of these classifiers by studying the impact of several parameters on the classification. We finish in Section (V) by discussing a potential direction for future works.

II. Classifiers

In this Section, we will describe the principle and the structure of all the basic classifiers used in this work: Decision Tree, k-Nearest Neighbor, Multi-Layer Perceptron and Support Vector Machine.

A. k-Nearest Neighbors

The k-Nearest Neighbor (KNN) is a cluster-based classification algorithm. That means there is no learning step [1], [2]. The algorithm tries to classify new entries into a cluster by comparing it with close neighbours. This method considers an element with similar properties to other elements close to the data space. Another hypothesis is that "normal" elements are located in a dense neighbourhood, and "abnormal" instances are far away from their closest neighbours.

To measure the similarity between data, we use a distance function. The most used function is the euclidean distance:

$$d(x,y) = \sqrt{\sum_{k=1}^{n}(x_k - y_k)^2} \tag{1}$$

Where:

- x_k is the k^{th} properties of x;
- y_k is the k^{th} properties of y;

The algorithm aims to find the K nearest neighbours of x in the data space. Then, the new element x is classified in a

class if most of the K's nearest neighbours are in this same class.

KNN does not use labelled data; it can be considered unsupervised learning.

The value of K is an important hyperparameter in KNN. It determines how many neighbours to consider when making predictions. A larger value of k means that the prediction will be based on a larger number of neighbours, which can lead to a smoother decision boundary but may also result in poorer accuracy. On the other hand, a smaller value of k means that the prediction will be based on a smaller number of neighbours, which can result in a more complex decision boundary but may also lead to overfitting.

B. Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP) is an Artificial Neural Network composed of several layers. An Artificial Neural Network is a computing system inspired by a human brain. It is built of several nodes connected to each other to make a network. Fig. (1) represents a typical MLP structure. There are three kinds of nodes:

- Input nodes
- Hidden nodes
- Output nodes

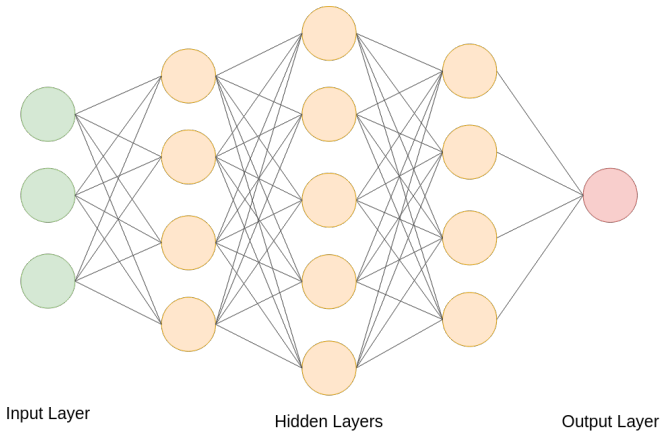

Fig. 1. Representation of an MLP

A node receives a signal, processes it, and can transmit the result to other connected nodes. Coefficients weigh the transmitted signals. The weighted signals are added as input by the neurons, and the node produces a result following an activation function based on this sum. This activation function can be compared to a logic gate. It determines if the node must be activated or not. There are several different activation functions. The most used are ReLu, Sigmoid, Linear function, and $tanh$ [3]. They are represented in Fig. (2).

The MLP can learn from incomplete, noisy or limited data. But they are sensitive to the number of input characteristics [4]. The more input characteristics, the more time the network needs to learn.

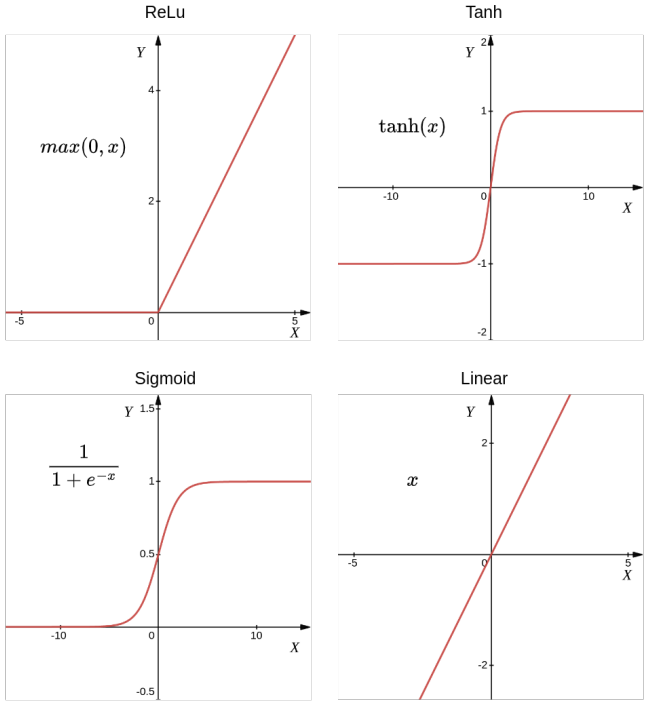

Fig. 2. Curves and equations of typical activation functions

C. Support Vector Machine

The Support Vector Machine (SVM) is based on the concept of margin and finding a separating hyperplane in the feature space between two classes [1]. The goal is to maximize the distance (called margin) between the hyperplane and the closest data points from each class.

The points on the margin are called support vectors, and the solution is based on their linear combination. Fig. (3). This approach is risk minimization instead of optimal classification.

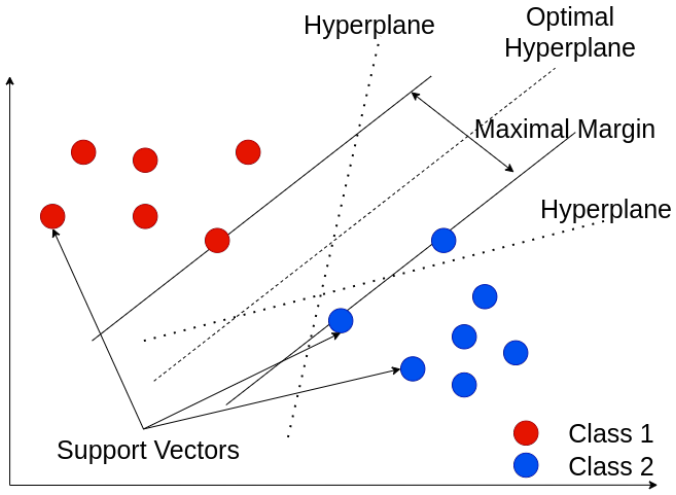

Fig. 3. 2D SVM representation

When it is impossible to separate points using a linear plane, SVMs use kernel functions ϕ to transpose them on higher-

dimension spaces [5]. Fig. (4). Indeed, using a non-linear kernel produce a bad classification.

SVM is very efficient to generalize and useful when the number of characteristics is high and the number of data elements is low.

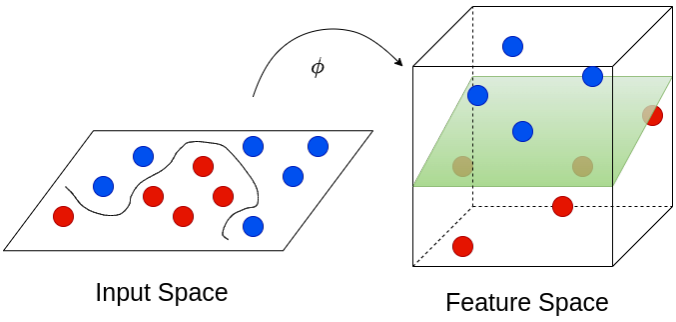

Fig. 4. Data transformation to a higher dimensions space

D. Decision Trees

A Decision Tree (DT) is a classifier that classifies data based on a decision sequence. These decisions are presented in a tree structure Fig. (5). Each root node or intermediate node represents an attribute, and each branch going down from this node represents the possible values this attribute can have. The leaves represent the classification categories.

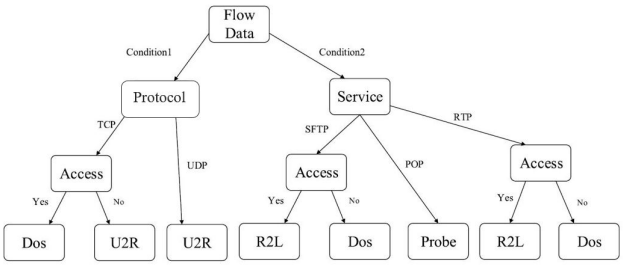

Fig. 5. Representation of a DT

There are several DT algorithms; we used the C4.5 algorithm in this paper [2]. It has the specificity that the chosen attribute has the highest information gain at each node.

III. Hybrid Adaptative Boosting algorithm

Our algorithm is a specific version of the AdaBoost [6], [7] algorithm and is a Boosting algorithm. A Boosting algorithm is based on combining several classifiers called *weaklearners* to build a robust classifier that improves the weak learners' performance. AdaBoost is an iterative process using errors made by a weak learner to improve the efficiency of the following weak learner. The idea behind AdaBoost is to iteratively reweight the training examples so that the next weak classifier focuses more on the examples that were misclassified by the previous weak classifiers. Fig. (6) represents the first weak learner (all data have the same weight). Fig. (7) represents the second weak learner where misclassified data have a higher weight.

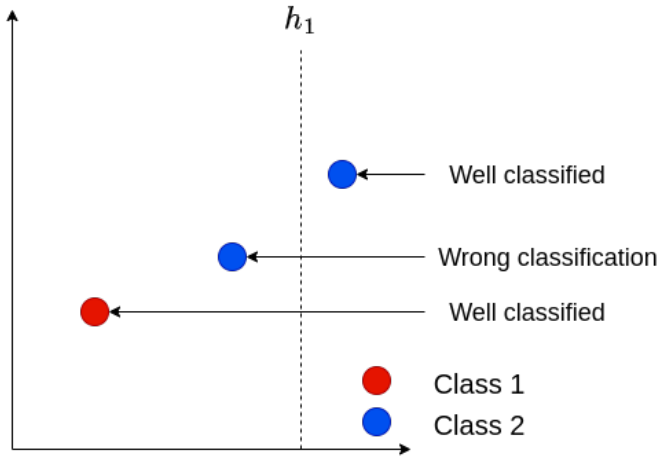

Fig. 6. AdaBoost first weak learner representation

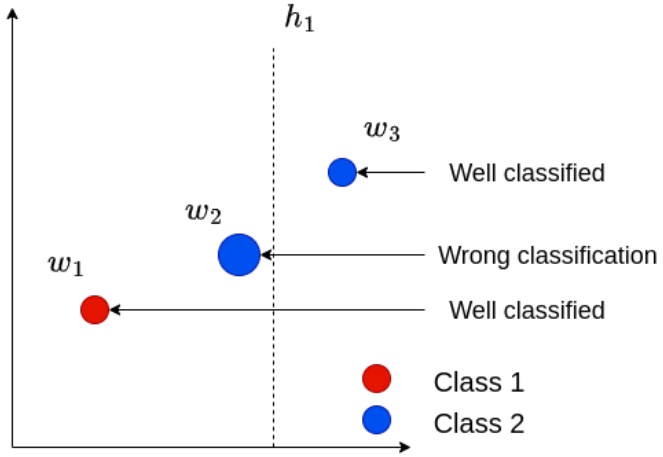

Fig. 7. AdaBoost second weak learner representation

Using this technic, the h_{t+1} weak learner will focus more on the areas in which the h_t learner was terrible.

A. Hybrid AdaBoost steps

This subsection describes in more details the different steps of the AdaBoost algorithm.

The first step is the attribute an initial weight to all elements in the training dataset:

$$w_0(x_i) = \frac{1}{n} \tag{2}$$

where n is the number of elements in the training dataset.

Then, a classifier (h_t) is trained on the whole training dataset before computing the weighted error. The computation of this error is done with the following equation:

$$\epsilon_t = \sum_{i=1}^{n} w_t(x_i)[h_t(x_i) \neq y_i] \tag{3}$$

y_i is the label of the data element i.

During the third step, the current model is assigned a weight useful for predicting new instances. This value is computed by using (4)

$$\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t}) \tag{4}$$

Finally, the weight of each element in the training dataset is updated before going to the next classifier. Indeed, as previously explained, the AdaBoost algorithm pays more attention, step by step, to the data element misclassified. The weight update is given by (5)

$$w_{t+1}(x_i) = \frac{w_t(x_i)}{Z_t} * e^{-\alpha_t * y_i * h_t(x_i)} \tag{5}$$

where

$$Z_t = \sum_{i=1}^{n} w_t(x_i) * e^{-\alpha_t * y_i * h_t(x_i)} \tag{6}$$

Z_t is a normalization factor used to ensure that the instances' weights follow a probability distribution.

After this update, the process is repeated on the next weak learner (h_{t+1}) beginning at the second step (classifier training) until all the weak learners are trained.

Each classifier is associated with a weight α to evaluate the method's performance on new data (testing data). Each classifier computes a prediction for the new data element (x_j). Because we are in a binary situation, the prediction can be -1 or 1. Each prediction is then weighted thanks to the learner corresponding α. All these weighted values are summed to determine the final prediction. This prediction can be written as follow:

$$H(x) = sign(\sum_{t=1}^{T} \alpha_t * h_t(x)) \tag{7}$$

AdaBoost usually works with Decision Tree as a weak learner. In our paper, we used the different algorithms previously described in this paper: Decision Tree, k-Nearest Neighbours, Multi-Layer Perceptron and Support Machine Vector.

IV. Parametric Study

This section presents the parametric study we performed on each classifier. The goal was to determine the parameters combination producing the best result for each learner. For this work, we used a training dataset of 23,415 PHP files containing 1,833 webshells. As mentioned earlier, these files had gone through a webshell detector with 5 agents. Each produces a value between 0 and 1 which is the probability this file is a webshell based on this specific criteria. The different agents of the detector are [8]:

- **Signature**: Based on file portion (usually malicious part) which can be used to identify it, or a similar one. If the file has been modified, the signature agent could not detect the file as malicious.
- **Fuzzy Hashing**: identify files that are similar to each other, even if they are not exact copies. It works by generating a fingerprint of a file that takes into account small variations and then comparing this fingerprint to the fingerprints of other files.
- **Dangerous routines**: mechanism checks if a PHP file uses some dangerous routines (*passthru, system*).
- **Obfuscation**: mechanism checks if the file uses non-ASCII characters, obfuscation functions (*base64decode,...*).
- **Entropy**: mechanism checks if a signal is expected or not. An unexpected signal may be significant, whereas an expected signal might be overlooked among a large number of similar signals.

We use a k-fold cross-validation method to evaluate the performance. The dataset is separated into k folds containing the same proportion of webshells. Training is done on $k-1$ folds and the testing is on the last one. This process is repeated k times with a different testing fold each time. The final result is the mean of all these intermediate results.

A. Performance evaluation criteria

The performance of a classifier is evaluated thanks to three statistical criteria:

- **AUC-ROC**: ROC is a graphical tool that illustrates the ability of a binary classifier by varying its discrimination threshold. The chart is built by plotting the true positive rate on x and the false positive rate on y. The AUC-ROC is the Area Under the Curve of a ROC. The closer to 1 the AUC is, the better the classifier.
- **AUC-PR**: Similar to AUC-ROC with a chart using Precision on x axis and Recall on y axis. This tool is more informative for an unbalanced dataset.
- **F-Measure**: metric that combines precision and recall. The closer to 1 the result is, the better the classifier.

During all our experimentation, we made the assumption the parameters are non-correlated (probably not perfectly correct). That allows us to test all parameters independently. We fixed all parameters to a specific value (generally based on what we have seen in the literature), and we varied the last one to identify the best value for this parameter. This process is repeated for each parameter of a classifier to determine the best parameters combination. Without this assumption, we should have tested all possible parameters combination, difficult in practice.

B. k-Nearest Neighbours parametric study

The first algorithm we studied is the KNN classifier. We consider 2 parameters in our parametric study:

- **k**: the number of neighbours used to classify the new element.
- **Distance type**: the method used to compute distance. We tested several methods: Euclidean, Chebyshev, Jansen-Shannon, Manhattan, Minkowski and correlation. In the case of using Minkwoski distance, an additional p parameter is required: the distance order.

To test the parameter k, we used the Euclidean distance as a fixed parameter and we varied k between 1 and 350. The results are in Fig. (8). The AUC-ROC and AUC-PR values stabilized around $k = 50$ with an AUC-ROC around 0.925. After this threshold, the computation time becomes more important for a negligible improvement.

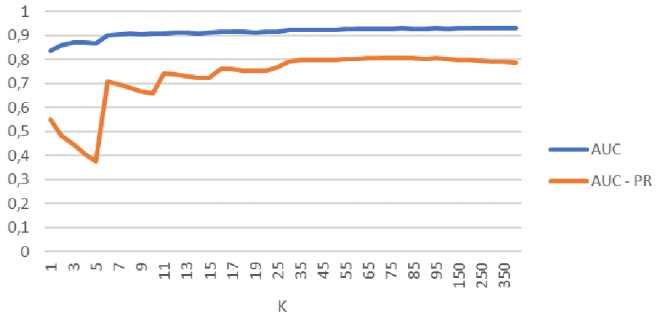

Fig. 8. Effect of Parameter k on AUC-ROC and AUC-PR. Fixed parameters: Distance: Euclidean

The distance computation method was tested with a k parameter fixed to 3. In Fig. (9) we observed a small improvement in AUC-PR with the Correlation method. This method is defined as [9]

$$Correlation(X,Y) = \frac{s_{x,y}}{s_x * s_y} \tag{8}$$

where $s_{x,y}$ is the covariance between X and Y:

$$s_{x,y} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y}) \tag{9}$$

and s_x and s_y are the standard deviation of X and Y.

The reason for the better results obtained with correlation probably lies in the fact that the calculation is based on the covariance between different features of the data. In contrast, for other distance measures, only the pairwise correspondence between different features is taken into account.

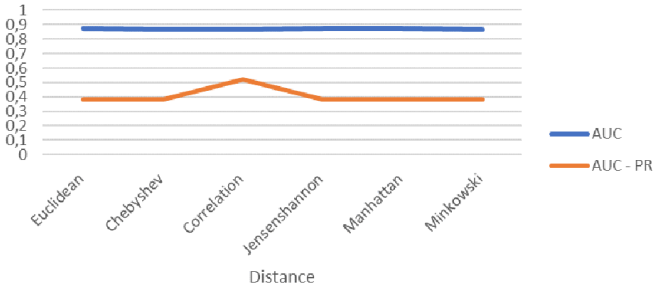

Fig. 9. Effect of distance computation method on AUC-ROC and AUC-PR. Fixed parameters: K: 3

Table (I) resumes the best value for each parameter with the associated AUC-ROC, AUC-PR and F-Measure.

The result obtained by combining the 2 optimal parameter values is less good than $k = 50$ and the Euclidean distance combination. The phenomenon can be attributed, in part, to the fact that the assumption that all parameters are independent of each other, which is made a priori, is likely to be incorrect.

TABLE I
KNN ALGORITHM PARAMETER VALUES PRODUCING BEST RESULTS DURING PARAMETRIC STUDY

Parameter	Value	AUC-ROC	AUC-PR	F-Measure
k	50	0,92473	0,800405	0,70488
Distance	Correlation	0,86963	0,51815	0,55901

C. Multi-Layer Perceptron parametric study

In this work, we used a very simple MLP architecture. The network contains only a single hidden layer with several neurons. An Artificial Neural Network has more parameters than the previous KNN algorithm. The set of these parameters is often called *Hyperparameters*. In our study we considered:

- **Activation function**: a function that introduces non-linearity into the output of a neuron. It determines whether a neuron should be activated or not based on the input signal it receives. The neuron's inputs are weighted and summed to be used as parameters for the activation function. In our work, we consider $tanh$, ReLu, sigmoid and a linear function. See Fig (2)
- **Neurons number**: the number of neurons in the hidden layer. Intuitively, we can think more neurons on the hidden layer produce a better result. In practice, too many neurons can lead to over-fitting.
- **Learning rate**: parameter that determines the step size or rate at which the model learns from the training data. It controls how much to adjust the model in response to the error calculated during training. A high learning rate may cause the model to overshoot the optimal solution, while a low learning rate may cause the model to converge slowly or get stuck in local minima.
- **Epochs number**: the number of times the algorithm will iterate over the entire dataset during training. Setting the right number of epochs is important as it affects the model's ability to generalize to new data. Too few epochs may result in underfitting, while too many epochs may lead to overfitting.
- **Batch size**: number of dataset elements processed by the network before updating its weights.

This parametric study was already treated in previous work [10]. But due to some technical limitations (the model was running in Google Colab), we were not able to test the complete range for all parameters initially planned. In this paper, we will only discuss the parameters whose value has changed because we are able to test our models on new values that were previously inaccessible to us. Table (II) contains the default parameter values used for our parametric study. These values were obtained from our previous work [10].

Our previous work tested the *epoch number* parameter between 100 and 350. It was not possible to run with values bigger than 350. This time, we tested this parameter from 350

TABLE II
DEFAULT PARAMETER VALUES USED IN PARAMETRIC STUDY

Parameter	Value
Neurons number	38
Learning rate	0.04
Batch size	2000
Epoch number	350
Activation function	ReLu

to 700. Fig. (10). We consider 350 as the optimal value for the rest of the paper. A higher epochs number value has a very small influence on AUC-ROC and AUC-PR for a computation time much higher.

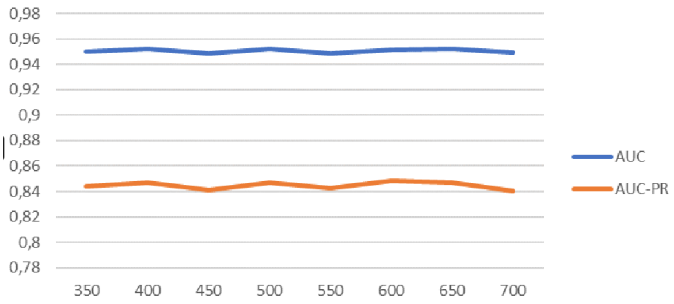

Fig. 10. Effect of Parameter *epochs number* on AUC-ROC and AUC-PR. Fixed parameters: Neurons number: 38 - Learning rate: 0.04 - Batch size: 2000 - Activation function: ReLu

The second parameter for which we were limited in our previous work is the *batch size*. We tested it from 1000 to 2000. This time, we varied the value from 2000 to 10.000. Fig. (11). The AUC-ROC and AUC-PR values decrease for a batch size bigger than 2000. The optimal value for the parameter *batch size* is 2000.

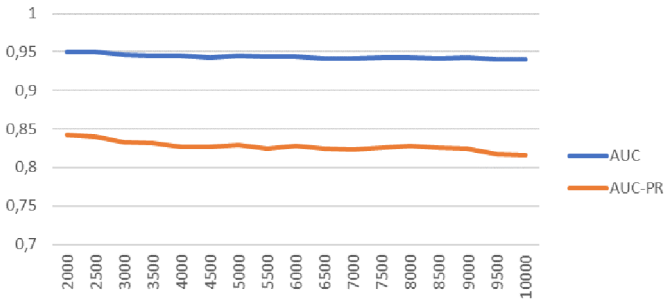

Fig. 11. Effect of Parameter *batch size* on AUC-ROC and AUC-PR. Fixed parameters: Neurons number: 38 - Learning rate: 0.04 - Epochs number: 350 - Activation function: ReLu

The set of optimal parameters is actually the optimal parameters computed in our last work and performance results are in Table (III).

We build a model using all these optimal parameters as *hyperparameters*. The combination of all the optimal parameters allows for obtaining excellent results, which are even better than those obtained through the various tests, as evidenced by Table (IV).

TABLE III
MLP PARAMETER VALUES PRODUCING BEST RESULTS DURING PARAMETRIC STUDY

Parameter	Value	AUC-ROC	AUC-PR
Neuron number	38	0,92473	0,80040
Learning rate	0.04	0,86963	0,51815
Batch size	2000	0.94963	0.84261
Epochs number	350	0.94957	0.84405
Activation function	ReLu	0.94989	0.83951

TABLE IV
PERFORMANCE INDICATORS WITH OPTIMAL PARAMETERS COMBINATION: NEURON NUMBER: 38 - LEARNING RATE: 0.04 - BATCH SIZE: 2000 - EPOCHS NUMBER: 350 - ACTIVATION FUNCTION: RELU

Criteria	Value
AUC-ROC	0.95753
AUC-PR	0.85940
F-Measure	0.79676

D. Support Vector Machine parametric study

As for the previous algorithms, we carried out a complete parametric study. The SVMs we tested have two specific parameters:

- **C**: corresponds to the cost of misclassification and determines how much the model should adjust to the training data. A higher value of C will lead to greater adaptation to the training data, but may also increase the risk of overfitting.
- **Kernel**: function used to transpose elements into a higher dimensional space. We tested two different kernel types: *Laplacian* and *polynomial*.

The kernel functions have their own specific parameters:

- **Sigma** (σ): specifies the non-linearity of the kernel function. The bigger σ will be, the more linear the decision will be. It is associated with Laplacian kernel
- **Polynomial degree**: associated with polynomial kernel.

Our parametric study can be divided into two sub-studies: one for Laplacian kernels, and the second for polynomial kernels.

1) Laplacian kernels: We tested the C parameter from 0.001 to 1000 with a σ value fixed to 0.1. Fig. (12) shows us the optimal value is somewhere between 10 and 200. But is difficult to be more accurate. The second test on Fig. (13) gives us 100 as the optimal value for C.

Just as for the determination of the optimal C parameter, the search for sigma required several attempts. Fig. (14) shows quite clearly a maximum around $\sigma = 0.1$. The second, Fig. (15) is less explicit but the value of 0.06 gives better results.

Table (V) contains the best parameter values and the associated performance indicator.

We combined the optimal parameters and obtained better results. These results are shown in Table (VI)

2) Polynomial kernels: For polynomial kernels, we varied the polynomial degree from 1 to 10 with C fixed to 3. Fig. (16) shows the AUC indicator seems to reach a maximum when the degree is 3, while for the AUC-PR indicator, the

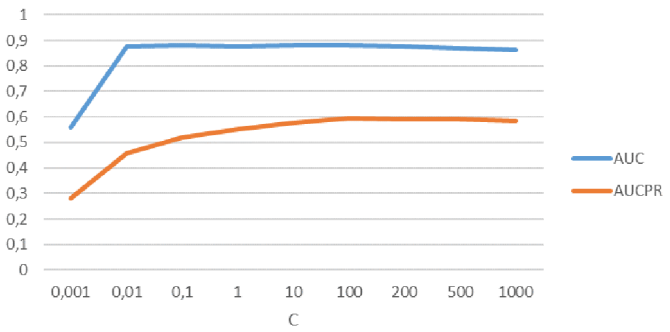

Fig. 12. Test 1 - Effect of Parameter C on AUC-ROC and AUC-PR. Fixed parameters: σ: 0.1

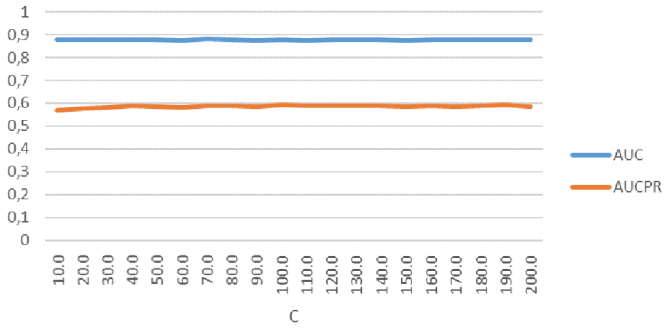

Fig. 13. Test 2 - Effect of Parameter C on AUC-ROC and AUC-PR. Fixed parameters: σ: 0.1

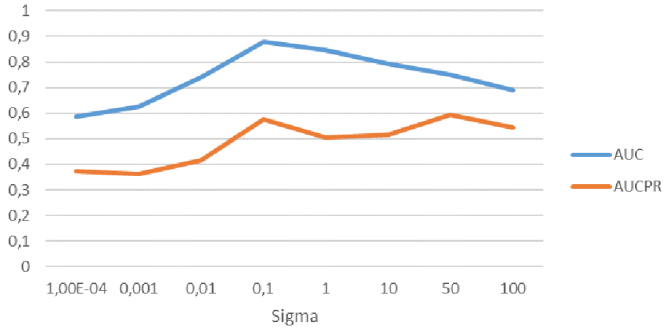

Fig. 14. Test 1 - Effect of Parameter σ on AUC-ROC and AUC-PR. Fixed parameters: C: 10

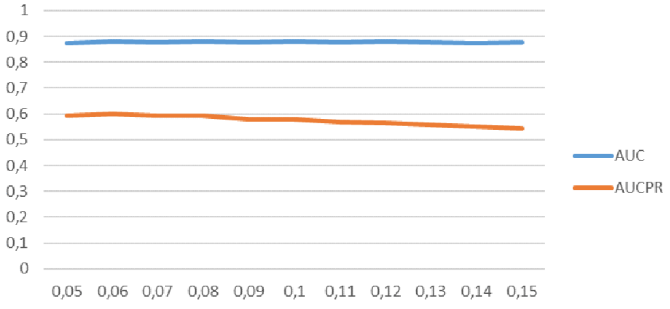

Fig. 15. Test 2 - Effect of Parameter σ on AUC-ROC and AUC-PR. Fixed parameters: C: 10

TABLE V
SVM LAPLACIAN ALGORITHM PARAMETER VALUES PRODUCING BEST RESULTS DURING PARAMETRIC STUDY

Parameter	Value	AUC-ROC	AUC-PR	F-Measure
C	100	0,87914	0,59422	0,74021
σ	0.06	0,88023	0,59879	0,74343

TABLE VI
PERFORMANCE INDICATORS WITH OPTIMAL PARAMETERS COMBINATION FOR SVM LAPLACIAN: C: 100 - σ: 0.06

Criteria	Value
AUC-ROC	0,84320
AUC-PR	0,61470
F-Measure	0,73899

more the degree increases, the more its value increases (0.55 for degree 3 to 0.62 for degree 10). The F-Measure follows the same trend as AUC-PR with an increase strongest between degrees 1 and 3 (from 0.63 to almost 0.7). This tendency would push us to take a degree as high as possible. The problem that can result from a (too) high degree is a longer learning time and possible over-learning. The choice of degree 5 seems to be a good compromise with satisfactory performance, the dimension of the data being also 5.

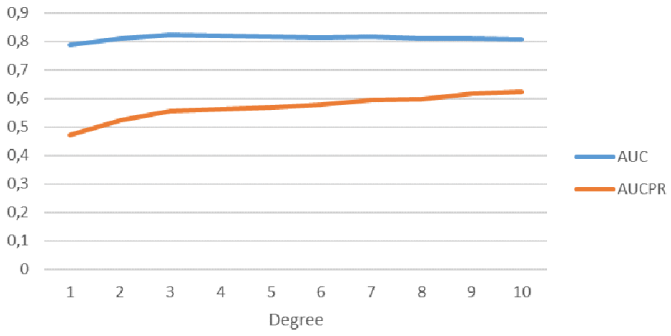

Fig. 16. Effect of Parameter polynomial degree on AUC-ROC and AUC-PR. Fixed parameters: C: 10

As for the Laplacian kernels, determining the optimal value of the parameter C required several attempts. With the first test on Fig. (17), we identify a working area of around 0.1. In the second test on Fig. (18), we can distinguish a really small maximum at 0.15.

Table (VII) contains the best parameter values and the associated performance indicator and Table (VIII) contains the result obtained for the model combining all the optimal parameters on polynomial kernels. We noted that the combination of optimal parameters gives less good results than some of the independent tests. It is likely because of our assumption of complete non-correlation between parameters.

We obtain better results with SVM using a Laplacian kernel.

E. Decision Tree parametric study

The last tested algorithm was the Decision Trees. This classifier is usually the *weaklearner* used in AdaBoost algorithm.

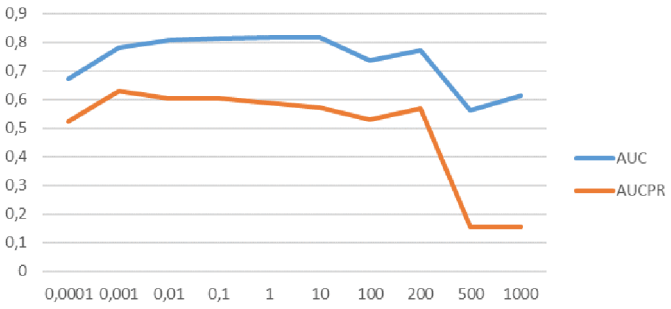

Fig. 17. Effect of Parameter *C* on AUC-ROC and AUC-PR. Fixed parameters: Polynomial degree: 5

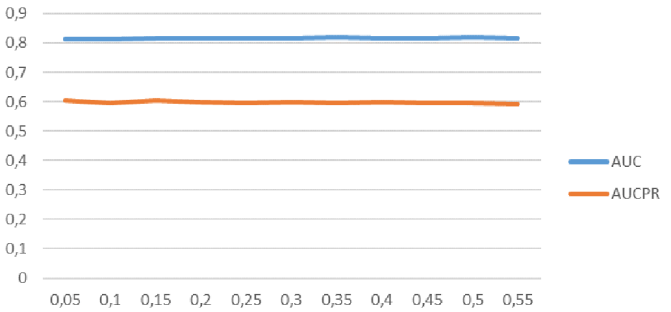

Fig. 18. Effect of Parameter *C* on AUC-ROC and AUC-PR. Fixed parameters: Polynomial degree: 5

Depending on the algorithm, DT can use several parameters. In our work we studied:

- **Pruning**: process of removing certain nodes or branches from the tree to improve its accuracy and reduce over-fitting. If true, pruning requires an additional parameter C: confidence factor. It specifies the minimum level of confidence required for a branch to be pruned. A value close to 1 is a more aggressive pruning.
- **Instance number**: defines the minimal number of instances by leaf. It is the minimum number of training instances required to form a leaf node.
- **Leaf separation**: leaf separation are exclusively binary or not.

TABLE VII
SVM POLYNOMIAL ALGORITHM PARAMETER VALUES PRODUCING BEST RESULTS DURING PARAMETRIC STUDY

Parameter	Value	AUC-ROC	AUC-PR	F-Measure
C	0.15	0,81672	0,60340	0,71767
Polynomial degree	5	0,81724	0,56838	0,69971

TABLE VIII
PERFORMANCE INDICATORS WITH OPTIMAL PARAMETERS COMBINATION FOR SVM POLYNOMIAL ALGORITHM: C: 0.15 - POLYNOMIAL DEGREE: 5

Criteria	Value
AUC-ROC	0,77446
AUC-PR	0,63499
F-Measure	0,69455

We varied the parameter C from 0 to 1 to determine its optimal value. Results are in Fig. (19). We observe better a result with $C = 0.6$. We also tried with non-pruned trees and obtained even better results.

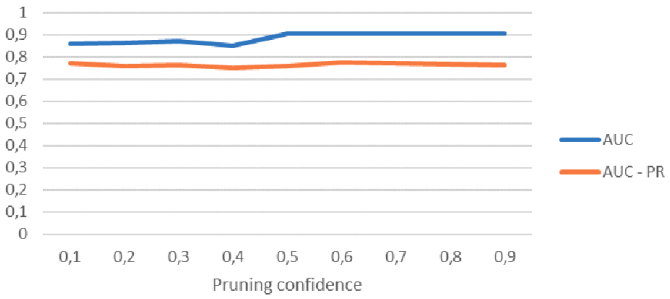

Fig. 19. Effect of Parameter *C* on AUC-ROC and AUC-PR. Parameters: pruning: true - Instance number: 2 - Separation leaf: non-binary

The second tested parameter was the minimal number of instances by leaf. The value varied from 2 to 100 and we obtained Fig. (20). We note a small maximum for the value 6.

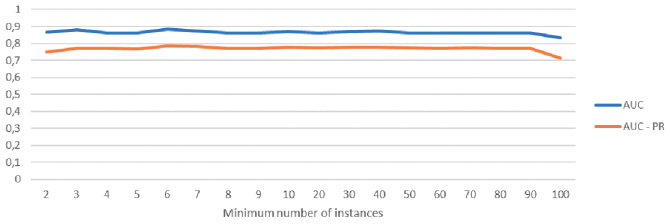

Fig. 20. Effect of Parameter *Instance number* on AUC-ROC and AUC-PR. Fixed parameters: pruning: true - C: 0.25 - Separation leaf: non-binary

We finally test if a DT gives better results if the leaf separation is exclusively binary or not. We obtained results a bit better with binary separation. Table (IX) summarizes all the optimal parameters and their associated AUC-ROC, AUC-PR and F-Measure. Table (X) contains the performance indicators for a Decision Tree combining all the optimal parameters. The result is better than during our previous tests.

TABLE IX
DECISION TREE ALGORITHM PARAMETER VALUES PRODUCING BEST RESULTS DURING PARAMETRIC STUDY

Parameter	Value	AUC-ROC	AUC-PR	F-Measure
Pruning	False	0,91499	0,78353	0,76806
Minimal instances	6	0,88269	0,78693	0,76102
Leaf separation	Binary	0,87058	0,75851	0,76228

F. AdaBoost Algorithm parametric study

Our last subsection is about a AdaBoost Algorithm composed of the four previously discussed classifiers.

We observed the influence of two main characteristics:

- **Classifier order**: the order in which the *"weak"* algorithms are used. Because of the structure of the AdaBoost

TABLE X
PERFORMANCE INDICATORS WITH OPTIMAL PARAMETERS COMBINATION FOR DECISION TREE ALGORITHM: PRUNING: FALSE - MINIMAL INSTANCES: 6 - LEAF SEPARATION: BINARY

Criteria	Value
AUC-ROC	0,91181
AUC-PR	0,80257
F-Measure	0,76723

algorithm we use, the order of its components has an influence on the final result. Indeed, during step $t+1$, the dataset elements are weighted according to the performance of the classifier in step t.

- **Optimal parameters**: the usage (or not) of the optimal parameters we determined during all this work. In the literature, AdaBoost uses *weaklearner*. It is interesting to test the performance of a Boosting algorithm made of several non-optimal classifiers.

1) Optimal Parameters: In this first test pool, we use a AdaBoost algorithm made of 4 classifiers: SVM, KNN, DT and MLP. As in previous tests, cross-validation was used. The results obtained vary quite significantly depending on the order in which the different models are considered. The order [SVM, MLP, KNN, DT] gives results in Table (XI) that are quite bad. Table (XII) shows the results for the order [MLP, SVM, KNN, DT] that is the best combination.

TABLE XI
PERFORMANCE INDICATORS FOR [SVM, MLP, KNN, DT] ADABOOST ALGORITHM WITH OPTIMAL PARAMETERS

Criteria	Value
AUC-ROC	0,43428
AUC-PR	0,05245
F-Measure	0,11601

TABLE XII
PERFORMANCE INDICATORS FOR [MLP, SVM, KNN, DT] ADABOOST ALGORITHM WITH OPTIMAL PARAMETERS

Criteria	Value
AUC-ROC	0,85635
AUC-PR	0,53897
F-Measure	0,69785

We also tried to combine several times the same classifier to build a new AdaBoost classifier. In the literature, AdaBoost is used with multiple Decision Trees as *weaklearners* [6]. We obtain the best result with an association of 4 SVM as mentioned in Table(XIII). The results are just lower than the best combination we made previously. The number 4 was chosen because the previous tests were done with 4 different classifiers.

2) Non optimal parameters: We perform the same analysis as with the optimal parameters but the classifiers are used without the optimal parameters. This analysis does not show a specific improvement or degradation of the result. Table (XIV) contains results for the best combination [KNN, DT, SVM, MLP]. We note the best combination with optimal parameters ([MLP, SVM, KNN, DT]) is the 18^{th} position in the list of results with non-optimal parameters with an AUC-ROC of 0.56604.

TABLE XIII
PERFORMANCE INDICATORS FOR [SVM, SVM, SVM, SVM] ADABOOST ALGORITHM WITH OPTIMAL PARAMETERS

Criteria	Value
AUC-ROC	0,83529
AUC-PR	0,59793
F-Measure	0,72365

TABLE XIV
PERFORMANCE INDICATORS FOR [KNN, DT, SVM, MLP] ADABOOST ALGORITHM WITH NON-OPTIMAL PARAMETERS

Criteria	Value
AUC-ROC	0,83965
AUC-PR	0,66418
F-Measure	0,76233

Our last analysis is the combination of 4 times the same classifier used without their optimal parameters. The best results are shown in Table (XV) and are obtained with [KNN, KNN, KNN, KNN] which was the worst combination with optimal parameters. And the worst with non-optimal parameters is [SVM, SVM, SVM, SVM] which was the best using optimal parameters. It is difficult to explain this result. One possible explanation is that KNNs are quite stable against resampling [11].

TABLE XV
PERFORMANCE INDICATORS FOR [KNN, KNN, KNN, KNN] ADABOOST ALGORITHM WITH NON-OPTIMAL PARAMETERS

Criteria	Value
AUC-ROC	0,80322
AUC-PR	0,45770
F-Measure	0,62280

Globally, combinations of several times the same classifier produce really bad results. Except for the values shown in this paper, all the other results are around or even above the random classifier value (AUC-ROC = 0.5).

V. CONCLUSION AND PERSPECTIVES

In this work, we tried to show to what extent a AdaBoost algorithm is more efficient or not than other more classical Machine Learning classifiers on the task of classifying a PHP file as a webshell or a harmless file. We performed a complete parametric study on four well-known classifiers: k-Nearest Neighbours, Multi-Layer Perceptron, Support Vector Machines and Decision Trees. We determined their optimal parameters and measure their individual performance with three criteria: AUC-ROC, AUC-PR and F-Measure.

The Multi-Layer Perceptron obtain the best result with an AUC-ROC of 0.95753 in its best configuration. It is important to note that the Decision Trees produce also quite good results: $AUC-ROC = 0.91181$ with a learning time around 350 times shorter than MLPs and KNNs give (at its best

configuration) $AUC - ROC = 0.92473$ with a computation time around 160 times shorter than MLPs.

The individual performance of the classifiers was quite good in general, but we thought about combining all these systems into an AdaBoost algorithm. Our goal was to improve the general performance of the classification. We used the 4 classifiers in several combinations and observe no improvements. The best result obtained by any version of our AdaBoost algorithm gives less good results than some of the individual classifiers.

The results obtained shed light on certain important elements regarding the AdaBoost method. As already explained above, although it involves the combination of several high-performing classifiers, the combination does not improve the results obtained individually. This underperformance can be explained by the fact that most of the combinations were performed on strong classifiers, or at least classifiers whose performance cannot be assimilated to weak learners, whereas in the literature, combinations are always performed on weak learners.

The result of this work can be consider as disappointing, but we observe generally good performance for the classification with singer classifiers. Moreover, we believe that highlighting something that is not working, is positive for the scientific community.

Nevertheless, several future perspectives are possible. We think it could be interesting to test this kind of classifier on another type of data (APTs, malware,...). In this work, we limited our tests to AdaBoost algorithm composed of 4 classifiers. We could try more different combinations or an AdaBoost algorithm with 10, 100 or even 1000 times weak learners as Decision Trees.

One another perspective for improvement could be to explore a specific *boosting* algorithm for KNNs. As mentioned earlier, KNNs are not very sensitive to an AdaBoost algorithm [11].

REFERENCES

[1] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *Expert Systems with Applications*, vol. 36, no. 10, pp. 11 994–12 000, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417409004801

[2] P. Mishra, V. Varadharajan, U. Tupakula, and E. S. Pilli, "A detailed investigation and analysis of using machine learning techniques for intrusion detection," *IEEE Communications Surveys Tutorials*, vol. 21, no. 1, pp. 686–728, 2019.

[3] V. Jain. Everything you need to know about "activation functions" in deep learning models. [Online]. Available: https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253

[4] H. Kim, J. Kim, Y. Kim, I. Kim, and K. Kim, "Design of network threat detection and classification based on machine learning on cloud computing," *Cluster Computing*, vol. 22, 01 2019.

[5] A. L. Buczak and E. Guven, "A survey of data mining and machine learning methods for cyber security intrusion detection," *IEEE Communications Surveys Tutorials*, vol. 18, no. 2, pp. 1153–1176, 2016.

[6] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002200009791504X

[7] T. Chengsheng, L. Huacheng, and X. Bing, "Adaboost typical algorithm and its application research," *MATEC Web of Conferences*, vol. 139, p. 00222, 01 2017.

[8] A. Croix, T. Debatty, and W. Mees, "Training a multi-criteria decision system and application to the detection of php webshells," pp. 1–8, 2019.

[9] V. Kotu and B. Deshpande, *Chapter 4 - Classification*, second edition ed. Morgan Kaufmann, 2019, pp. 65–163. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128147610000046

[10] A. Croix, T. Debatty, and W. Mees, "Comparison of a supervised trained neural network classifier and a supervised trained aggregation."

[11] N. García-Pedrajas and D. Ortiz-Boyer, "Boosting k-nearest neighbor classifier by means of input space projection," *Expert Systems with Applications*, vol. 36, no. 7, pp. 10 570–10 582, 2009. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417409002140