A Framework for Classifying Web attacks while respecting ML Requirements*

Nourhène BEN RABAH[1] and Ines BEN TEKAYA[1]

Centre de Recherche en Informatique, Université Paris 1 Panthéon Sorbonne Paris, France
Nourhene.Ben-Rabah | Ines.Ben-Tekaya @univ-paris1.fr

Abstract. Injection and Cross Site Scripting attacks are among the ten critical security risks to web-based applications. It is difficult, to provide a complete signature for firewalls that detect such attacks. Therefore, there are several proposals based on Machine Learning (ML) methods capable of detecting various web attacks from evolutive, heterogeneous data at large scale, without the need for expert knowledge. Unfortunately, web attacks detection have been addressed only from a ML algorithm viewpoint, there is a lack of clarity regarding the quality and amount of the training data, the hyperparameters tuning and the evaluation method. Low and poor data quality may compromise the success of the most powerful ML methods. Additionally, it is easy to build a model that is perfectly adapted to the dataset but unable to generalize the new unseen data. This paper introduces F2MW, a framework for multi-classifying web attacks with respect to the ML requirements.

1 Introduction

According to the Open Web Application Security Project, injection attacks and Cross Site Scripting (XSS) attacks are among the ten critical security risks to web servers and web-based applications. The injection attacks such as SQL injection occur when untrusted data is sent to a code interpreter as part of a command or query [11]. XSS flaws occur whenever an application with untrusted data sends it to a web browser without proper validation or escaping [9].

Existing security solutions are generally based on firewalls, which filter web application inputs using a predefined set of rules, and signature-Intrusion Detection Systems (IDS), which detect attacks using a set of signatures. Unfortunately, it is difficult to identify a complete rule for firewalls and full signatures for IDS. In addition, because of the diversity of attacks, these systems cannot detect zero-days attacks.

Given these limitations, there are several proposals based on ML methods that are capable of distinguishing various web attacks from evolutive, heterogeneous data at a large scale, without the need for expert knowledge [8]. Unfortunately, in these works, web attacks detection was addressed only from a ML

* Supported by Centre de Recherche en Informatique.

algorithm viewpoint. However, the performance of a given ML method does not depend only on the algorithm. There are other interesting pre-emptive requirements against potential overfitting and degradation of the model's performance, such as the quality and the amount of the training data, the hyperparameters tuning and the evaluation method. Low quality data may compromise the success of the most powerful ML methods such as ensemble methods and deep learning methods. The goal of this paper is to introduce a framework for multi-classifying web attacks while respecting ML requirements. To reach this overall goal, we defined two Research Questions (RQs):

- **(RQ1).** How to make raw network data available for any ML algorithm? To answer this question, we present a data pre-processing phase based on three steps: data cleaning, data transformation and feature selection. This phase has a heterogeneous input network data that can be noisy, scattered, incomplete and constantly evolving. It has as output clean training data, containing only the most relevant features selected through mutual information measure. Using this measure, we can detect any kind of relationship between data of different type (continuous or discrete). In addition, it is insensitive to the size of datasets.
- **(RQ2).** How to prevent the overfitting problem when there is an imbalance in classes of the training set? In answering this question, we exploit the stratified cross validation which is a powerful premptive data resampling method against potential overfitting and degradation of the model performance.

These various issues are addressed through a framework called F2MW. Experimental results on the benchmark CICIDS2017 dataset shows that our proposal addresses the problems of unbalanced classes and overfitting while offering high performance (i.e detection accuracy).

The remaining of this paper is organized as follows: Section 2 describes the F2MW, while section 3 presents two strategies that validate our framework. On Section 4, we discusses the use of ML approaches on IDS, before presenting the conclusion drawn from this work on section 5.

2 F2MW: A framework for multi-classifying web attacks

This section presents a framework for multi-classifying web attacks while respecting ML requirements. The elaboration process comprises three major phases: Data pre-processing, Hyperparameter tuning and Evaluation.

2.1 First phase: Data pre-processing

This subsection presents the first phase of our framework, in which the data collection, data cleaning, data transformation and features selection are depicted.

Data collection. The purpose of data collection is to identify the dataset used to evaluate our framework. As a large numbers of datasets are presented

in the literature, we have seek to identify a dataset that is publicly accessible, large in size, describes different normal user behaviors types and different most up-to-date web attacks, where each instance is labelled. For this reason, we use a recent IDS dataset namely CICIDS2017 [16] from the Canadian Institute for cybersecurity, which covers normal user behaviours and six attack profiles. Also, it is publicly. Each attack profile is presented in a CSV file including instances describing normal user behaviors and others describing different intrusion scenarios for each attack profile. To validate our framework, we use the sub-set describing only web attacks and for which there is a class unbalance problem. It will be noted CICIDS2017 Web. It contains 170366 instances including 168186 instances describing normal user behaviors (labelled 'Benign'), 1507 instances describing Web Brute Force attacks (labelled 'Brute Force'), 652 instances representing Cross-Site Scripting attacks (labelled 'XSS') and 21 instances describing SQL Injection attack (labelled 'SQL injection').

Data cleaning. It was designed to apply a normalization on some features to obtain a [0,1] range. Indeed, some features in the CICIDS2017 Web dataset vary between [0,1] while other features vary between [0,∞). For normalization, we use equation 1 defined as follows:

$$x_i = \frac{x_i - x_{min}}{x_{max} - x_{min}} \tag{1}$$

There are some rows in a given dataset, which have the feature 'Flow Bytes' equal to 'Infinity' or 'NaN'. We cannot utilize this feature because of its values; To resolve this concern, we have removed all the rows.

Data transformation. Typically in ML, we have to process two data types: quantitative data and qualitative data. Some classifiers can work directly with qualitative data such as the k-nearest neighbors (KNN), Naive Bayes (NB), Decision Trees (DT), etc. However, many require the inputs and outputs variables to be digital in order to operate properly. Therefore, it is always essential to convert qualitative data into numerical data. In our dataset, each instance is represented by numerical data (i.e. the 78 features) and categorical data (i.e. feature no. 79). Categorical data must be converted into numerical data. Therefore, the purpose of data transformation is to assign an integer value for each class in this way: if label is 'benign' then it will be '0', if label is 'brute force' then it will be '1', if label is 'SQL Injection' then it will be '2' and if label is 'XSS' then it will be '3'.

Feature selection: The purpose of feature selection is to identify the most relevant features of the 'CICIDS2017 Web' dataset. Therefore, we use the Mutual Information (MI) since it is a measure capable of detecting any sort of relationship between datasets, besides being insensitive to the size of the datasets [15]. We propose a formal specification to our selection feature problem based-on generic cases for X and Y where X is a feature and Y is a label.

Case 1: (Continuous, Discrete). In the first case, X is represented by the continuous type and Y by the discrete type. For example, X can be flow duration and Y can be benign or attack. The MI is obtained by equation 2:

Proof. We have X is a continuous variable and Y a discrete variable. We can prove that $I(X,Y) = H(X) + H(Y) - H(X,Y)$

if X is a discrete variable and Y continuous variable we have: [15]

$$I(X,Y) = H(X) + H(Y) - H(X,Y) \tag{2}$$

$$Since\ I\ is\ symmetric => I(X,Y) = I(Y,X) \tag{3}$$

$$Since\ the\ join\ entropy\ is\ symmetric\ [10] => H(X,Y) = H(Y,X) \tag{4}$$

(2), (3) and (4) $=> I(X,Y) = H(X) + H(Y) - H(X,Y)$ when X is a continuous variable and Y a discrete variable.

Case 2: (Discrete, Discrete). In the second case, X is represented by the discrete type and Y by the discrete type. For example, X can be the destination port and Y can be benign or attack.

The MI of two jointly discrete random variables X and Y is calculated as a double sum. The MI is obtained by equation 5 from [5]:

$$I(X,Y) = \sum_{y \in Y} \sum_{x \in X} p_{(X,Y)}(x,y) \log \left(\frac{p_{(X,Y)}(x,y)}{p_X(x) p_Y(y)} \right) \tag{5}$$

2.2 Second phase: Hyperparameter tuning

Random Forests (RF) are ensemble learning methods. They combine the predictions of several models that are built from the same algorithm which is the DT. Therefore, a RF is a set of DTs that is trained on different subsets of the training set, which are selected randomly, in order to have diversified results. To determine the class of a new observation, it is enough to obtain the predictions for each of the trees and then choose the class with the highest number of votes. The random selection of the training set subsets can be done with or without replacement [4].

A hyperparameter is a parameter of the learning algorithm and not of the model. Adjusting the hyperparameters of an algorithm enables finding the optimal hyperparameters of a given dataset. The RF has several hyperparameters that can influence its performance [14]. They are depicted in Table 1. It is well known that in most cases, RF works reasonably well with the default values of the hyperparameters specified in software packages. Nevertheless, adjusting the hyperparameters can improve their performance. To choose an excellent combination of values for our hyperparameters, one solution is to build a model for each possible combination of considered hyperparameter values (see Table 1). Therefore, it is necessary to test (10*3*3*2*10*2) possible values. This method is expensive and slow. To overcome these limits, we conducted a random search method that consists of evaluating a given number of combinations, noted n. The principle of this method is described by the algorithm 1.

Table 1. Description of the RF hyperparameters and the considered values.

Hyperparameter	Description	Considered values
Number of trees	The number of trees in the forest.	200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000
Min _split	The minimum number of samples required to split an internal node.	2, 5, 10
Min_leaf	The minimum number of samples required to be at a leaf node.	1, 2, 4
Max_features	The number of features to consider when looking for the best split.	Max, sqrt
Max_depth	The maximal depth of the trees, which is the maximal number of splits until the terminal node.	10, 20, 30, 40, 50, 60, 70, 80, 90, 100
Bootstrap	The random selection of the subsets of the training set can be done with or without replacement.	With and without replacement

Algorithm 1: Hyperparameters tuning using random search

Result: Best combination of hyperparameters values

for $i \leftarrow 0$ **to** n **do**
  for each hyperparameter, select a random value among the considered values;
  Divide the dataset E into 10 stratified fold; `/* (E0, E1,..., E9) */`
  for $j \leftarrow 0$ **to** 9 **do**
    $Test \leftarrow E_j$; `/* Forms the test set */`
    $Train \leftarrow E - E_j$; `/* Forms the training set */`
    Train the algorithm on the training set; `/* Returns a model */`
    Use the resulting model to predict the labels of the test set; `/* Returns a set of predicted labels */`
    Calculate the average accuracy by comparing the set of predicted labels with the set of real labels in the test set; `/* average accuracy is detailed in the subsection 3.1 */`
  end
  Calculate the overall accuracy of the 10 folds ;
end
Select the best combination of values that gives the best overall accuracy;

2.3 Third phase: Evaluation

Cross Validation (CV) [20] is one of the most widely used data resampling methods used to estimate the classifier's error rate, adjust its parameters and prevent overfitting. It uses the entire dataset for learning and validation. It divides it into roughly equal K-folds. Each of the K-folds is used as a test set and the rest of the K-1 folds are used for the classifier's training. For each test fold, the classifier's error rate is calculated by comparing the predicted labels with the real labels of the test fold instances. The error rate for K-folds is the average of the values computed in the loop. K-folds CV is a powerful data resampling method against overfitting. However, since it is usually randomly prepared, imbalanced test and train sets can be obtained; therefore; leading to a degradation of the model performance. For example, our dataset contains 170366 instances of 4

classes: n_{Benign}=168186, $n_{BruteForce}$=1507, n_{XSS}=652 and $n_{SQLInjection}$=21. If CV is performed, it is quite possible that some test sets contain only benign instances (or Brute Force instances, etc.), since there is a disproportionate ratio of instances in each class.

To deal with this problem, we focused on the stratified CV [17, 18] which is a variation of K-folds CV that splits the folds, thus allowing each to contains the same distribution of classes to ensure that relative class frequencies is approximately preserved in each train and test fold. Details about this method are provided in the validation strategies section.

3 Validation strategies

This section presents two strategies to validate our framework. After verifying that stratified CV addresses the problems of unbalanced classes and overfitting, a verification is conducted with the goal of ensuring that the feature selection based on MI measure actually improves the classification accuracy. To do this, we set the best combination of hyperparameters values of RF using the stratified CV and then we set the number of neighbours, since it is a hyperparameter of the MI measure.

3.1 Performance metrics

A much better way to evaluate the performance of a classifier is to compute the confusion matrix. Each column of the matrix represents the instances in a predicted class, while each row represents the instances in a real class. Confusion matrix is represented through four basic measures: True Positive (TP) which refers to the normal instances that are correctly predicted, True Negative (TN) which refers to the web attacks instances that are correctly predicted, False Negative (FN) which refers to the normal instances that are incorrectly predicted and False Positive (FP) which refers to web attacks instances that are incorrectly predicted. For an effective intrusion detection system, (TP) and (TN) rates must be maximized, and (FP) and (FN) should be minimized.

In our work, we propose a framework for multi-classifying web attacks. Table 1 presents the multi-class confusion matrix of our proposal. It illustrates the TP_{Benign}, FN_{Benign}, FP_{Benign} and TN_{Benign} of the benign class. On the diagonal of the matrix, we can see the TP_{Benign} and the TN_{Benign} (green boxes) that represent the well classified instances, either in normal operation or in case of attacks. The FP_{Benign} represent attacks which are classified as normal operation (blue boxes). FN_{Benign} represent normal instances which are classified as attacks, i.e. false alarms that are triggered without the occurrence of any real attacks (orange boxes). Misclassified attacks MIS_{Benign} represent attacks which are classified as other attacks (red boxes).

Although the confusion matrix provides us with a lot of information about the quality of our framework, we calculate more concise parameters, such as precision, recall and accuracy. The macro-average precision ($Precision_{MA}$), the

	Benign	Brute Force	Sql Injection	XSS
Benign	TP_{Benign}	FN_{Benign}	FN_{Benign}	FN_{Benign}
Brute Force	FP_{Benign}	TP_{Benign}	MIS_{Benign}	MIS_{Benign}
Sql Injection	FP_{Benign}	MIS_{Benign}	TP_{Benign}	MIS_{Benign}
XSS	FP_{Benign}	MIS_{Benign}	MIS_{Benign}	TP_{Benign}

Fig. 1. Multi-class confusion matrix to illustrate TP_{Benign}, FN_{Benign}, FP_{Benign} and TN_{Benign} of the benign class.

macro-average recall ($Recall_{MA}$) and the average accuracy ($Accuracy_A$) of all classes are calculated respectively through these equations:

$$\text{Precision}_{\text{MA}} = \frac{\sum_{i=1}^{l} \frac{\text{TP}_\text{i}}{\text{TP}_\text{i}+\text{FP}_\text{i}}}{l} \tag{6}$$

$$\text{Recall}_{\text{MA}} = \frac{\sum_{i=1}^{l} \frac{\text{TP}_\text{i}}{\text{TP}_\text{i}+\text{FN}_\text{i}}}{l} \tag{7}$$

$$\text{Accuracy}_{\text{A}} = \frac{\sum_{i=1}^{l} \frac{\text{TP}_\text{i}+\text{TN}_\text{i}}{\text{TP}_\text{i}+\text{FN}_\text{i}+\text{FP}_\text{i}+\text{TN}_\text{i}}}{l} \tag{8}$$

With l is the number of classes.

3.2 Stratified cross-validation against the problems of unbalanced classes and overfitting

To determine how well our framework is able to solve the problems of unbalanced classes and overfitting, we calculated the average accuracy of RF using 10-folds CV and 10-folds stratified CV. For the experiments, we used the Scikit-learn [13] ML library and we assign for the RF algorithm the default hyperparameters values defined by this library. These values are described in Table 1. Table 2 depicts the $Accuracy_A$, $Precision_{MA}$ and $Recall_{MA}$ per-fold using 10-fold CV and 10-fold stratified CV. These results show that the overall $Accuracy_A$ is improved after using the CV. The gain is about 0,22%. With 10-fold cross-validation, the model seems to have a good predictive accuracy, according to the average accuracy of about 99,32%, as opposed to the present case.

	Benign	Brute Force	Sql Injection
Benign	17024	0	0
Brute Force	8	1	4
Sql Injection	0	0	0

Fig. 2. The multi-class confusion matrix of the first split using 10-kfold CV.

	Benign
Benign	17036

Fig. 3. The multi-class confusion matrix of the 8^{th}, 9^{th} and 10^{th} splits using 10-kfold CV.

Table 2. Performance comparison per-fold using 10-fold CV and 10-fold stratified CV.

	RF with hyperparameters default values and 10-fold CV			**RF with hyperparameters default values and 10-fold stratified CV**		
	$Accuracy_A\%$	$Precision_{MA}\%$	$Recall_{MA}\%$	$Accuracy_A\%$	$Precision_{MA}\%$	$Recall_{MA}\%$
Fold 1	99,92	66,65	35,89	99,48	54,85	52,18
Fold 2	98,86	66,66	55,34	99,53	77,38	64,72
Fold 3	98,99	66,65	56,12	99,50	51,88	51,78
Fold 4	99,33	66,65	56,95	99,60	81,48	68,24
Fold 5	96,57	66,62	33,89	99,57	79,01	65,94
Fold 6	99,57	49,96	28,01	99,51	75,95	62,69
Fold 7	99,99	50	49,97	99,54	52,54	51,68
Fold 8	100	100	100	99,63	82,66	68,61
Fold 9	100	100	100	99,60	80,94	66,69
Fold 10	100	100	100	99,49	50,21	49,23
Overall measure	99,32 +/- 1	73,32 +/-1,8	61,62 +/- 26	99,54 +/- 0,05	68,69 +/- 13	60,18 +/- 7,5

Unbalanced classes. We examine the confusion matrix of the first fold (Fig. 2) that represents an average accuracy of 99.92%. It looks like a very good predictive accuracy, but this is not. The fold has a $Precision_{MA}$ of only $66,65\%$ that matches [$Precision_{Benign}$= $99,95\%$, $Precision_{BruteForce}$=100%, $Precision_{XSS}$= 0%] and a $Recall_{MA}$= $35,89\%$ that matches [$Recall_{Benign}$ =100%, $Recall_{BruteForce}$=$7,60\%$, $Recall_{XSS}$ = 0%]. The model seems to have a good predictive accuracy simply because benign class is very well classified thanks to the sufficient number of instances of this class, so that $Precision_{Benign}$ and $Recall_{Benign}$ have a large values. However, brute force and XSS classes are poorly classified because of the limited number of instances in the fold. In addition, the model is not able to predict the SQL injection attacks since the first split (the training and the test sets) does not contain any instances of this class. This explains why the confusion matrix does not represent this attack class.

Overfitting: We also study the confusion matrix of the 8^{th}, 9^{th} and 10^{th} splits (Fig. 3) that represent an average accuracy of 100%. These splits contain only instances of benign class. For this reason, the model is overfitting the data and has no generality.

To address the above-mentioned problems of unbalanced classes and overfitting, we use the 10-stratified CV method. Each split contains the same distribution of classes to ensure that relative class frequencies are approximately preserved in each train and test fold. Figures 6 and 5 show the confusion matrices of the first split and the 8^{th} split, respectively. All classes are present in the matrices. The benign class is very well classified. The model makes 0 false alarms (orange boxes). Also, the brute force class is well classified. However, the SQL injection class and XSS class are poorly classified because the fact that XSS attack is highly similar to brute force. This explains why at least 33 XSS instances are classified as brute force attack.

	Benign	Brute Force	Sql Injection	XSS
Benign	16819	0	0	0
Brute Force	11	107	0	33
Sql Injection	3	0	0	0
XSS	7	33	1	25

Fig. 4. The multi-class confusion matrix of the first split using 10-kfold stratified CV.

	Benign	Brute Force	Sql Injection	XSS
Benign	16818	0	0	0
Brute Force	0	129	0	21
Sql Injection	1	0	1	0
XSS	7	33	1	25

Fig. 5. The multi-class confusion matrix of the 8^{th} split using 10-kfold stratified CV.

3.3 From hyperparameters tuning and feature selection to evidence that the framework learns better

To determine the improvement of our framework learning capacity, we set the best combinations of hyperparameters values of RF using the algorithm 1. After setting the number of combination (denoted n) to *100*, the RF is trained *10* folds for each of *100 candidates*, thus totalling *1000* fits in order to determine the best combination of values. Table 3 depicts the $Accuracy_A$, $Precision_{MA}$ and $Recall_{MA}$ per-fold for the model representing the best combination of hyperparameters which are: Number of trees=1600, Min-split=5, min-leaf=1, Max-features=78, Max-depth=10 and bootstrap= "without replacement". The results show that the average accuracy is improved. The gain is about 0,04%.

Table 3. The performance of the best model.

	RF with best combinaison of hyperparameter and 10-fold stratified CV		
	$Accuracy_A$%	$Precision_{MA}$%	$Recall_{MA}$%
Fold 1	99,53	57,77	50,32
Fold 2	99,64	92,93	64,98
Fold 3	99,58	82,94	64,08
Fold 4	99,60	80,13	63,97
Fold 5	99,54	76,02	63,35
Fold 6	99,55	49,90	49,60
Fold 7	99,58	62,34	63,64
Fold 8	99,60	60,30	51,58
Fold 9	99,60	53,70	50,70
Fold 10	99,63	80,75	76,75
Overall measure	99,58+/-0,3	69,69+/-13	69,90

To further improve the average accuracy, our framework is based on a feature selection method using MI. The first important question that arises when applying this method is how many features should be used to obtain a good accuracy. Therefore, our main goal is to determine whether the increase in this number enhances or degrades the performance of our framework. The second question that should be answered concerns the number of neighbours, since it is an hyperparameter of the MI measure. The two above-mentioned questions can be solved experimentally by carrying out a study on the variation of the number of features and the number of neighbors. Table 4 presents the obtained features according to their numbers. Figure 6 illustrates the results of this study.

As can be observed, the best average accuracy is obtained with 9 features and 10 neighbors.

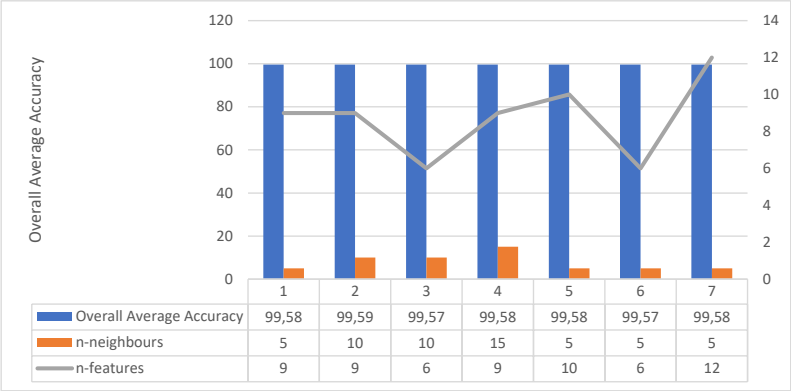

Fig. 6. Overall average accuracy according to the number of features and the number of neighbors.

Table 4. The obtained features according to their numbers.

Number of features	Obtained features
6	Flow Packets/s, Fwd IAT Mean, Fwd IAT Min, Fwd Header Length, Fwd Packets/s, Init_Win_bytes_backward
9	Flow Packets/s, Flow IAT Mean, Flow IAT Std, Fwd IAT Mean, Fwd IAT Min, Fwd Header Length, Fwd Packets/s, Fwd Header Length.1, Init_Win_bytes_backward
12	Flow Packets/s, Flow IAT Mean, Flow IAT Std, Flow IAT Max, Fwd IAT Mean, Fwd IAT Std, Fwd IAT Min, Fwd Header Length, Fwd Packets/s, Bwd Packets/s, Fwd Header Length.1, Init_Win_bytes_backward

4 Related Works and discussion

In our study, we are rather interested in web attacks detection based on supervised ML. We have therefore considered various research works for our literature review including the ML and cybersecurity communities. Instead of focusing on specific conferences, we have performed keywords-based queries like “web attacks”, “attacks detection” and “supervised learning” while precising the research area, like “ML” or “cybersecurity”. We have discarded papers published before 2013 as we wanted to focus on recent works. These keyword-based queries returned more than 100 research papers. We read their abstracts and selected

those which presented web attacks detection systems using supervised learning. Among the remaining articles, we focused on the approaches that seemed appropriate in terms of scalability.

Following this methodology, we finally selected 15 papers that we analyzed in depth. Among those, we may note, for instance, in [1], the authors suggest a binary classification based on Support Vector Machine (SVM), KNN and DT algorithms and report that KNN provide better performance than the other algorithms. In [12], the authors propose an IDS based on Artificial Neural Network (ANN) using only single hidden layer. They use the CICIDS2017 dataset to validate the performance of their proposal. They selected 8 types of attacks of which XSS and SQL injection are not included to avoid facing the unbalanced classes problem.

To obtain better performance, several studies use ensemble methods such as [6] that built a model for IDS using RF classifier. Nevertheless, the authors use RF with the default hyperparameters of Weka software. In [2], the authors propose a binary and a multi-class classifier on based on XGBoost algorithm. The method was compared to other classifiers such as AdaBoost, NB, MLP and KNN. It achieves a good accuracy rate of about 99.54% for multi-class classification. Nevertheless, the authors do not use a feature selection method that can further improve the performance of their method. In [3], the authors present a data dimensionality reduction schema for IDS using ensemble and standalone classifiers. To test the performance of their approach, they use algorithms such as XGBoost, CTtree, SVM and Net without hperparameters tuning. Thus, they study the performance of these algorithms in binary classification. For this reason, they are not faced with the problem of unbalanced classes.

To extract relevant features without effort and improve their performance, several studies use different deep neural networks architectures [7, 21, 19]. For example, in [21], the authors present a deep learning approach for intrusion detection using Recurrent Neural Network (RNN). They study the performance of the resulting model in multi-class classification. For evaluation, they compare the performance of their proposal to the NB, RF, Multilayer Perceptron (MLP) and SVM. However, there was no hyperparameters tuning to the algorithms. They only tuned the number of neurons and the learning rate in their method. The same problem is also investigated in [19], where the authors use MLP and the CICIDS2017 dataset to detect web attacks. Their solution offers a good accuracy of 98.8% using 5 layers. However, it does not identify the name of the web attack (SQL injection or XSS) since they group these two attacks in the same class called 'Web'. However, it does not identify the name of the web attack (SQL injection or XSS) since they group these two attacks in the same class called 'Web'.

Our literature review allows us to make several observations. Most authors who are not specialists in ML used the algorithms without mentioning the hyperparameter tuning phase. Another observation we can make is that many works don't address the problem of class imbalance. Besides, we can note that deep learning solutions are becoming more and more popular. Indeed, one of their

strengths is that they allow skipping the feature selection step. However, these approaches need a normalization step since since numerical data will have a negative inuence on gradient descent optimizing methods, with a smaller learning rate if they do not have similar ranges of values.

5 Conclusion

In this paper, we propose F2MW to detect web attacks while respecting ML requirements. Three dimensions are considered that respond to the following questions: How to make raw network data available for any ML algorithm? How to prevent the overfitting problem when there is an imbalance in classes of the training set? As way of validation of our framework, we present the stratified CV method and feature selection based on MI measure that improves the classification accuracy. In our future work, we aim to test the F2MW framework with various types of network attacks on an Internet of Things environment to further enhance its performance.

References

1. Aksu, D., Ustebay, S., Aydin, M.A., Atmaca, T.: Intrusion detection with comparative analysis of supervised learning techniques and fisher score feature selection algorithm. In: 32nd International Symposium on Computer and Information Sciences. pp. 141 – 149. Springer, Poznan, Poland (2018)
2. Bansal, A., Kaur, S.: Extreme gradient boosting based tuning for classification in intrusion detection systems. In: International Conference on Advances in Computing and Data Sciences. pp. 372–380. Springer (2018)
3. Bansal, A., Kaur, S.: Data dimensionality reduction (ddr) scheme for intrusion detection system using ensemble and standalone classifiers. In: International Conference on Advances in Computing and Data Sciences. pp. 436–451. Springer (2019)
4. Breiman, L.: Bagging predictors. Machine learning **24**(2), 123–140 (1996)
5. Cover, T.M., T.J.: Elements of Information Theory. Wiley ed. (1991)
6. Farnaaz, N., Jabbar, M.: Random forest modeling for network intrusion detection system. Procedia Computer Science **89**, 213–217 (2016)
7. Kaur, S., Singh, M.: Hybrid intrusion detection and signature generation using deep recurrent neural networks. Neural Computing and Applications pp. 1–19 (2019)
8. Khraisat, A., Gondal, I., Vamplew, P., Kamruzzaman, J.: Survey of intrusion detection systems: techniques, datasets and challenges. Cybersecurity **2**(1), 20 (2019)
9. Kozik, R., Choraś, M., Renk, R., Hołubowicz, W.: A proposal of algorithm for web applications cyber attack detection. In: IFIP International Conference on Computer Information Systems and Industrial Management. pp. 680–687. Springer (2015)
10. Michael, A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge University Press (2010)
11. OWASP, T.: Top 10-2017 the ten most critical web application security risks. URL: owasp. org/images/7/72/OWASP_Top_10-2017_% 28en **29** (2017)
12. Park, S., Park, H.: Ann based intrusion detection model. In: Workshops of the International Conference on Advanced Information Networking and Applications. pp. 433–437. Springer (2019)

13. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
14. Probst, P., Wright, M.N., Boulesteix, A.L.: Hyperparameters and tuning strategies for random forest. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **9**(3) (2019)
15. Ross, B.C.: Mutual information between discrete and continuous data sets. PloS one **9**(2) (2014)
16. Sharafaldin, I., Lashkari, A.H., Ghorbani, A.A.: Toward generating a new intrusion detection dataset and intrusion traffic characterization. In: ICISSP. pp. 108–116 (2018)
17. Sundarkumar, G.G., Ravi, V.: A novel hybrid undersampling method for mining unbalanced datasets in banking and insurance. Engineering Applications of Artificial Intelligence **37**, 368–377 (2015)
18. Takase, T., Oyama, S., Kurihara, M.: Evaluation of stratified validation in neural network training with imbalanced data. In: 2019 IEEE International Conference on Big Data and Smart Computing. pp. 1–4. IEEE (2019)
19. Vinayakumar, R., Alazab, M., Soman, K., Poornachandran, P., Al-Nemrat, A., Venkatraman, S.: Deep learning approach for intelligent intrusion detection system. IEEE Access **7**, 41525–41550 (2019)
20. Wong, T.T.: Performance evaluation of classification algorithms by k-fold and leave-one-out cross validation. Pattern Recognition **48**(9), 2839–2846 (2015)
21. Yin, C., Zhu, Y., Fei, J., He, X.: A deep learning approach for intrusion detection using recurrent neural networks. IEEE Access **5**, 21954–21961 (2017)