Cyber Threat Monitoring Systems - Comparing Attack Detection Performance of Ensemble Algorithms

Eva Maia¹, Bruno Reis¹, Isabel Praça¹, Adrien Becue², David Lancelin², Samantha Dauguet Demailly², and Orlando Sousa¹

¹ GECAD - Research Group on Intelligent Engineering and Computing for Advanced Innovation and Development, School of Engineering of the Polytechnic of Porto (ISEP), Porto, Portugal
{egm,missr,icp}@isep.ipp.pt
² Airbus Defence and Space, Paris, France

Abstract. Cyber-attacks are becoming more sophisticated and thereby more difficult to detect. This is a concern to all, but even more to Critical Infrastructures, like health organizations. A Cyber Threat Monitoring System (CTMS), providing a global approach to detect and analyze cyber-threats for health infrastructures is proposed by combining a set of solutions from Airbus CyberSecurity with a machine learning pipeline to improve detection and provide awareness from cyber side to a more global approach that will combine them with physical incidents. The work is being carried out in the scope of SAFECARE project. In this work, we present the CTMS architecture and present our experimental findings with ensemble learning methods for intrusion detection.

Several parameters of six different ensemble methods are optimized, using Grid Search and Bayesian Search approaches, in order to detect intrusions as soon as they occur. Then, after the determination of best set of parameters for each algorithm, the attack detection performance of these six different ensemble algorithms using the CICIDS 2017 dataset are calculated and discussed. The results obtained identified Random Forest, LightGBM and Decision Trees as the best algorithms, with no significant difference in the performance, using a 95% confidence interval.

Keywords: Machine Learning · Intrusion Detection System (IDS) · Ensemble learning · Parameters optimization · Random Forest · Decision Trees · LightGBM · Adaboost · Rusboost · CICIDS2017

1 Introduction

Over the last decade, the European Union has faced numerous threats that quickly increased in their magnitude, changing the lives, the habits and the fears of hundreds of millions of citizens. The sources of these threats have been heterogeneous, as well as weapons to impact the population. Health services are
at the same time among the most critical infrastructures and the most vulnerable ones. They are widely relying on information systems to optimize organization and costs, whereas ethics and privacy constraints severely restrict security controls and thus increase vulnerability. The aim of SAFECARE project is to bring together the most advanced technologies from the physical and cyber security spheres to achieve a global optimum for systemic security and for the management of combined cyber and physical threats and incidents, their interconnections and potential cascading effects. SAFECARE cyber security solutions include a IT threat detection system, an Advanced file analysis system, a threat detection system for Building Monitoring Systems (BMS) and a E-health device security analytics system, all monitored in an overall Cyber Threat Monitoring System (CTMS) that feeds a data exchange layer, where all physical and cyber security incidents are analysed in a combined way through an impact propagation model. These detected incidents are then available through a Threat Response and Alert system, providing awareness to different stakeholders, from SOC operators to National health agency, Police, Firefighters, etc. In this paper, we will describe the CTMS and detail our experimental findings in using ensemble techniques for intrusion detection. The main objective of the IT Threat detection system is to improve network traffic incident/threat detection and investigation. Machine Learning methods have been widely used for this analysis since it can understand the behavior of an attack using known traffic datasets, and then can detect attacks in the network. The first experimental findings we will share in this paper rely on the application of Ensemble learning techniques. Ensemble learning [4] is a machine learning paradigm where multiple models are trained independently to solve the same problem, and then combined to get better results. As they often outperform single models for many types of problems [16], this technique has been widely used in intrusion detection [8,11,15]. Six different ensemble algorithms were considered for this study. In order to determine the best set of parameters for each ensemble algorithm, different parameter optimization techniques (Grid Search and Bayesian optimization) were applied. Then, after the determination of best set of parameters for each algorithm, their attack detection performance was calculated using a macro averaged score of the recall, precision and F1-score metrics. This way it is possible to understand the performance of each algorithm but also the trade-offs between precision and recall, so important in the attack detection field. The results showed that Random Forest and LightGBM are the best algorithms with no significant difference in the performance. The rest of the algorithms can be classified according with its attack detection performance in the following order: Decision Tree, Rusboost, Balanced Random Forest and Adaboost.

2 Cyber Threat Monitoring System

One of the main outcomes of the SAFECARE project is a cyber threat monitoring system (CTMS) that aims at improving the detection of Advanced Persistent Threats (APTs) [2] and zero-day attacks on IT and BMS systems, as
shown in Fig. 1. The e-health device analytics solution collects data from medical devices, combines it with other (public) data sources and performs analytics to derive meaningful security data to help identify, assess and manage threats and risks affecting the e-health devices. The advanced file analysis system performs an in-depth analysis of the files extracted from network traffic by the IT and BMS threat detection systems, and the cyber threat monitoring systems receives all security events produced by the IT and BMS threat detection system. The advanced file analysis system detects malicious files based on different approaches: static analysis such as signature matching, heuristic analysis and dynamic analysis [3]. Signature matching is a deterministic method only effective if the malware is already known. Heuristic methods are able to identify several variants of a virus but can generate false-positive events. Dynamic analysis consists in sand boxing the file in a suitable environment in order to highlight suspicious behaviours. IT threat detection system supports the objective of improving incident detection by providing network monitoring and producing relevant information. The IT threat detection system captures the network traffic and performs near-real time analysis in order to detect suspicious behaviour and scale up security events to the cyber threat monitoring system. The IT threat detection system uses a hybrid approach combining both non-supervised methods and supervised methods in order to improve threat investigation and threat detection on the network traffic. The current solution is based on the network threat detection engine Suricata and is associated with a correlation engine Graylog. A machine learning module includes different techniques combined in a hybrid approach. In this paper we study, analyze and compare the performance of Ensemble Techniques and our experimental findings based on the training with public datasets.
3 Ensemble Methods in Intrusion Detection

Ensemble models are particularly adept at dealing with large datasets due to the ability to partition the data to train different classifiers with different samples and combining their results. Diversity is also essential since it allows classifiers to correct each other’s errors. Random Forest, Adaboost, Rusboost and LightGBM are some of the most used ensemble methods in intrusion detection systems. Decision trees are the basic method of all these ensemble methods.

Stefanova and Ramachandran [20] proposed a two-stage classifier for network intrusion detection. Experimental tests led the authors to state that the approach presented is a superior method compared to the existing data mining models in network security, since the time for performing the analysis is relatively short, and the accuracy is remarkable.

Due to its nature, intrusion detection evaluation datasets are composed by imbalanced data where the proportion of attack events across all data is not evenly distributed. Random Forest is more robust than other widely known methods when considering imbalanced datasets. Even then, in some situations, the imbalance may influence the accuracy. Balanced Random Forest improves the ability of Random Forest models deal with imbalanced data [1].

Adaboost has been employed in several intrusion detection approaches but it is most commonly used in signature detection. Mazini et al. [14] proposed a hybrid solution where the Artificial Bee Colony (ABC) algorithm is used to optimize the search for the best feature space (feature selection) and Adaboost. M2 (a multiclass adaboost [23]) is used in a multiclass classification setting. In order to validate their results, the authors used False Positive Rate (FPR), recall and accuracy on the NSL-KDD and ISCXIDS2012 datasets. It was concluded that the proposed solution outperformed other methods with 99.61% detection rate, 0.01 FPR, and 98.90% accuracy.

Latah and Toker [13] presented a comparative study on the choice of an efficient anomaly-based intrusion detection method. The authors focused on supervised machine learning approaches, using several typical classifiers, such as decision trees, bagging trees, AdaBoost, Rusboost, etc. Using the well-known NSL-KDD dataset and based on experimental studies, the authors concluded that decision trees approach shows the best performance in terms of accuracy, precision, F1-score, area under the curve and McNemar’s test. In addition, approaches like bagging trees, AdaBoost and Rusboost outperformed other conventional machine learning methods with a confidence level over 99.5%. Yulianto et al. [22] improved AdaBoost-based IDS Performance on CICIDS 2017 Dataset. The method proposed by the authors outperforms the performance of previous works with the accuracy of 81.83%, precision of 81.83%, recall of 100%, and F1 Score of 90.01%.

4 Datasets

Data are the most valuable asset to develop an efficient intrusion detection system. In the literature, exist several public available datasets that are intended
to resemble real data traffic, such as KDD-99 [5], DARPA 98/99 [21] and ISCX2012 [19]. In this work we will use the CICIDS2017 dataset [6], which was created to overcome the issues of existing datasets. This dataset is available in eight different (csv) files, containing five days of normal and 3 days of intrusive traffic.

Despite ensemble learning techniques are capable of handling large datasets, we have decided to sample the dataset for training and testing purposes, since it is more efficient and cost-effective than surveying the entire dataset. Thus, 30% of the data were chosen using a stratified sampling method, i.e., the new dataset had the same proportion of attacks and benign traffic as the complete dataset. Nevertheless, this proportional cut affected attacks with less representation, namely infiltration, heartbleed, and SQL injections, which had less than 40 instances each. To solve this problem it was decided to include, all these attacks in the new dataset. It is important to note that this addition changed the original distribution of traffic, and can create an undesired bias in the data. However, this bias it would be similar to the bias present in any other oversampling or undersampling method. Moreover, since we are only resample a small amount of instances (around 100 in total) the bias is even less significant. Therefore, it can be said that the dataset was sampled twice, in a first instance it was sampled to 30%, using a stratified method, making the underrepresented classes almost nonexistent; in a second instance all the underrepresented classes were included. Lastly, before starting the training phase, the new dataset was partitioned in train (77%) and test (23%) sets using again a stratified sampling strategy.

5 Ensemble Optimization

Most machine learning algorithms have several parameters that should be adjusted properly, otherwise the selected algorithm will not achieve optimal results. Several studies have been successfully proposed in parameter optimization to obtain the most accurate classification models [7]. In this paper, we mainly used three different methods: Grid Search, Bayesian Search and manual tuning. Grid Search is an exhaustive search based on defined subsets of the parameter space. Ramadhan et al. [18] applied the Grid Search method for tuning parameters in the well-known classification algorithm Random Forest. Also in this work, we have tuned Random Forest, Rusboost and Decision Tree parameters using Grid Search. The process was done as follows: first, a range to test the parameters was chosen by examining previous works and estimating the boundaries; second, a Grid Search was applied using the boundaries and some values in between. This process was accompanied by a 3-fold cross-validation to ensure that the results were stable in the training data and did not depend on random chance. k-fold cross-validation is a re-sampling procedure used to evaluate machine learning models, increasing the consistency and the quality of the results. In this procedure the training test is divided into k non-overlapping parts of equal proportion (randomized). Following this division k models are trained, each using its part as a validation set, and the other k − 1 parts as a
training set. Finally, all k models’ performance is averaged to give a final result without overfitting the test set.

A common alternative to Grid Search is Bayesian Optimization which could be employed when the number of parameters and, consequently, the computational cost of doing a Grid Search are high. Bayesian optimization is an iterative algorithm. In each iteration, a probabilistic surrogate model is fitted to all observations of the target function made so far. Then, an acquisition function, which uses the predictive distribution of the probabilistic model, determines the utility of different candidate points, trading off exploration and exploitation [9]. We will use Bayesian optimization to tune LightGBM parameters. In the following sections we will describe the tuning of each individual algorithm as well as the parameters that are being tuned.

5.1 Random Forest and Balanced Random Forest

The optimization of parameters for Random Forest and Balanced Random Forest are very similar. Moreover the parameters that needs to be tuned are exactly the same:

- **Number of trees**: number of trees that are part of the Random Forest;
- **Max depth**: maximum depth of each tree;
- **Max features**: number of features that should be used to train each tree;
- **Splitting Criterion**: the criterion to test the quality of the splits.

Random Forest is a bagging method that ensembles several decision trees. So, as for any other bagging method, random samples of training sets (bootstrap samples) for each Random Tree are produced. The number of trees is then an important parameter to analyse. Thus, we built two plots (Fig. 2). As can be seen, the number of trees does not increase the F1-score beyond a small number of trees, in this case only about 100 trees are needed for performance improvements to be halted. This behavior is expected and in accordance with the literature [17]. According to the results, we can conclude that if F1-score was the only performance metric of interest any number of trees greater than 50 would be an acceptable choice. However, in the case of anomaly detection, execution time is also a concern. By analyzing the plot on the right (see Fig. 2), it is possible to

![Fig. 2. Plot of F1-score (left) and execution time (right) against the number of trees.](image)
Fig. 3. Plot of F1-score (left) and execution time (right) against the maximum depth of the trees.

Fig. 4. Plot of F1-score (left) and execution time (right) against maximum number of features.

conclude that time and number of trees are positively correlated, which suggests, together with the graph on the left, that the ideal number of trees necessary to reach a plateau in performance could be any number above 50 trees. However, to account for the inherent randomness of the process 100 trees will be considered.

Another important parameter to tune is the depth of each individual tree, which is the only mechanism that Random Forest has to control the bias. Decision trees when grown sufficiently deep have relatively low bias. However, this can bring high variance, since the learned parameters, such as the structure of the decision tree, will vary considerably with the training data. The problem is that the model learns not only the actual relationships in the training data,

Fig. 5. Plot of F1-score (left) and execution time (right) against the splitting criteria.
but also any noise that is present. In Random Forest, this variance is controlled 
by the bagging of the various trees, which means that by averaging out each 
prediction it is possible to reduce the error of the model on the test data, 
comparatively to the error of a single tree. Thus, the entire forest will have lower 
variance but not at the cost of increasing the bias. As can be seen in Fig. 3, 
the performance improves with tree depth, and execution time stays somewhat 
stable for each value. This indicates that growing trees to the maximum depth 
(until all leaves are pure) has a small computation cost but can bring good 
performance improvements. In the Random Forest only a subset of all features are 
considered for splitting each node in each decision tree. Therefore, the maximum 
number of features to use in every single split is one of, if not the most, important 
parameter in a Random Forest. With the analysis of the Fig. 4 it is possible to 
observe that the performance plateaus after 70%. This means that the algorithm 
could only produce meaningful splits with a high number of features probably 
due to a large amount of noise variables. This type of values point to the need 
to select the more relevant features.

Finally, a not so important parameter when building a Random Forest is 
the splitting criteria. Although the small importance for Random Forest due its 
ensemble nature, split criterion is a fundamental issue in decision trees. There 
are two main criteria: Gini Impurity and Information Gain (entropy). The Gini 
Impurity represents the probability that a randomly selected sample from a 
node will be incorrectly classified according to the distribution of samples in the 
node [12].

Changes in splitting criteria parameter rarely cause significant performance 
differences in Random Forest. Figure 5 shows that using Gini Impurity results 
slightly lower performance and slightly longer training time. In this case entropy 
was chosen, but the choice of Gini Impurity would not significantly affect the 
final results. Therefore, the final parameters chosen were:

- **Number of trees**: 100;
- **Max depth**: None;
- **Max features**: 70%;
- **Splitting Criterion**: Entropy.

Note that the **Max depth** parameter is None in order to expand the nodes 
to the maximum depth.

### 5.2 Decision Tree

In the case of decision trees, the parameters are very similar to those of Random 
Forest: **Max depth, Max features** and **Splitting Criterion**. This happens 
because the parameters used in Random Forest focus on controlling each individu-
ual tree instead of controlling the interaction between them. As such the 
descriptions will be similar to those used in the previous section.

In Fig. 6, it is easily observable that the maximum F1-score occurs when the 
maximum depth is about 17. Training time stabilizes when the maximum depth
approaches 10. This means that, contrarily to Random Forest choosing a higher maximum depth will not increase training time. However, there is no mechanism for reducing the variance of deep trees, which can make deep trees to outperform training data and to perform poorly on test data. As such, deep trees are not recommended and a depth of around 17 should be enough to meet performance needs.

Figure 7 presents the graphs for maximum number of features. As in Random Forest case, it is possible to notice two plateaus in the both functions, one between 40% and 60% and another between 80% and 100% features. This suggests a high number of noise variables, which indicates that using a low number of features, such as 10% or 20%, would highly contaminate the samples, leading to bad splits and consequently to poor results.

Interestingly, the best splitting criterion in decision trees is not the same as Random Forest. As can be observed in Fig. 8, although the training time is very similar, the criterion that leads to the best F1-score is Gini Impurity instead of entropy, with a 2% increase. Despite it seems very negligible, it can make a difference by reducing metrics like false positives and false negatives, that already have very low values. Thus, the final parameters for decision trees are:

- Max depth: 17;
- Max features: 80%;
- Splitting Criterion: Gini Impurity.

5.3 Adaboost and Rusboost

Adaboost and Rusboost, due to its nature, have almost the same parameters that can be tuned:

- Max Number of estimators: number of trees to fit in the model (can be reduce by early stopping);
- Learning rate: Rate of convergence of evaluation metric (higher converges faster);
- Max depth: depth of each boosted tree;
- Sampling strategy: which classes to undersample (exclusive to rusboost).
Fig. 7. Plot of F1-score (left) and execution time (right) against maximum number of features.

Fig. 8. Plot of F1-score (left) and execution time (right) against splitting criteria.

Fig. 9. Plot of F1-score (left) and execution time (right) against number of estimators.

The number of estimators of a boosting model is one of the most important parameters. It represents the number of trees in the forest. Usually the higher the number of trees the better to learn the data. However, adding a lot of trees can slow down the training process considerably. Moreover, the number of estimators or trees needs to be balanced with the learning rate to tune the model.

In Fig. 9, it can be seen that for a given learning rate (0.05) the F1-score seems to increase monotonically when the number of trees is increased. This means that the F1-score increases rapidly in the early stages and stabilizes later. Therefore, choosing the number of trees is not so important as long as it is not too small or too high (may result in overfitting). A good starting point would be around 100 trees.
Another extremely important parameter is the learning rate or shrinkage. This parameter can be interpreted as the contribution scale of each tree to the prediction. This means that when there is a low shrinkage the model takes longer to converge, and as such will need more trees. Therefore, the learning rate and the number of trees are often seen together, as a tradeoff: when one is decreased the other must be increased, or the model is at risk of overfitting. A good way to ensure stability is to choose a low learning rate value and then select the number of trees. Another reason for adjusting shrinkage first is the computational cost. Usually, the algorithm looks for the convergence point when training. Thus, if the shrinkage is selected beforehand, the algorithm can select an optimal number of trees, reducing the computational cost.

**Fig. 10.** Plot of F1-score (left) and execution time (right) against learning rate.

Figure 10 shows that shrinkage does not improve after 0.05, due to the low number of estimators (100), that limits the extent to which the convergence can be delayed. If the number of trees was increased, the learning rate would be expected to decrease. In terms of training time the model has erratic computational costs with expected monotonic trends when randomness is accounted for.

**Fig. 11.** Plot of F1-score (left) and execution time (right) against max depth.
Normally, the depth of the boosted trees is not a concerning fact when training a boosting algorithm. Tree stumps (trees with one root and two leaves) are often good models for weak learners and rarely need adjustment. When using trees deeper than one, there is a risk of increasing the variance, since models make fewer mistakes in each iteration. Thus, it is hard to train subsequent models using the misclassified samples of the previous model. Therefore, Hastie et al. [10] recommend a depth between 3 and 7, and mention that rarely more is needed and may result in a significant variance addition. As can be seen in Fig. 11, the model seems to stabilize the depth around 5. However, the score would be expected to fall if the depth continued to increase more than the chart boundaries. Finally, the training time follows a similar function and plateaus the depth around 5.

![Fig. 12. Plot of F1-score (left) and execution time (right) against sampling strategy.](image)

In the case of RusBoost there is one more parameter to take into account: the sampling strategy. Sampling strategy dictates which of the classes will be undersampled in each iteration of the boosting algorithm. There are 4 options for undersampling: undersample the majority class, all classes but not the majority class, all classes but not the minority class and finally it is possible to undersample all classes. In the case of intrusion detection, it is intuitive to want to undersample the majority class and leave the few samples of attack without any sampling. However, in the left plot of Fig. 12, it is possible to observe that the most successful undersampling technique is not majority as expected but undersampling every single class. This type of sampling is unexpected and as such would need a more thorough study. Finally, when it comes to the computation cost undersampling every class but the not majority displays the worst performance with around 200% more training time when compared to undersampling every class. Therefore, in this case the final parameters are:

- **Max Number of estimators**: 150;
- **Learning rate**: 0.05;
- **Max depth**: 5;
- **Sampling strategy**: all.
5.4 LightGBM

In the case of gradient boosting machines like LightGBM the tuning of parameters is not as straightforward as in the algorithms previously described. The solution is use Bayesian optimization, which uses an underlying gaussian process as a heuristic to find the parameters that optimize a black box model, i.e., it constructs a posterior distribution of functions (gaussian process) that best describes the function that needs to be optimized. There are many parameters that can be used in Bayesian optimization, nonetheless, the most important ones are: the number of iterations and the number of random explorations. The first dictates how many steps the model will execute to try to find the best parameters and the second tries to prevent the process from getting stuck in local minima by exploiting random solutions.

To find the best number of boosted trees a LightGBM model with 600 max trees was ran using 10-fold cross-validation together with an early stopping parameter of 200 iterations. It was possible to conclude that in all folds the model did not improve after 350 trees. As such, this number was chosen as an immutable parameter in the optimization process, along with other LightGBM parameters such as the objective function as *multiclass*, the number of classes as 15, class weight as *balanced* and bagging frequency as 5.

The parameters eligible for optimization were:

- **Bagging Fraction**: percent of data to sample (a value of 1 means all the data are used at each iteration, so no bagging is used), ranging from 0.8 to 1;
- **Feature Fraction**: fraction of features to select in each iteration, ranging from 0.1 to 0.9;
- **Max depth**: depth of each boosted tree, ranging from 5 to 9;
- **Number of leaves**: maximum number of leaves of each boosted tree, ranging from 50 to 80 leaves;
- **Learning rate**: rate of convergence of evaluation metric (higher converges faster), ranging from 0.001 to 0.1.

Table 1 presents the results of Bayesian optimization for these parameters. The line with the best results is highlighted.

| Iteration | F1  | Bagging Fraction | Feature Fraction | Learning Rate | Max Depth | Number of Leaves |
|-----------|-----|------------------|------------------|--------------|-----------|----------------|
| 0         | 0.918 | 0.875          | 0.861          | 0.073        | 7.0       | 55.0           |
| 1         | 0.908 | 0.811          | 0.146          | 0.086        | 7.0       | 71.0           |
| 2         | 0.914 | 0.804          | 0.876          | 0.083        | 6.0       | 55.0           |
| 3         | 0.778 | 0.808          | 0.162          | 0.005        | 9.0       | 50.0           |
| 4         | 0.932 | 1.0            | 0.9            | 0.1          | 9.0       | 80.0           |
| 5         | 0.576 | 1.0            | 0.9            | 0.001        | 5.0       | 80.0           |
| 6         | 0.712 | 0.8            | 0.1            | 0.001        | 9.0       | 63.0           |
| 7         | 0.737 | 1.0            | 0.9            | 0.001        | 9.0       | 75.0           |
| 8         | 0.923 | 1.0            | 0.9            | 0.1          | 5.0       | 66.0           |
| 9         | 0.923 | 1.0            | 0.9            | 0.1          | 5.0       | 50.0           |
| 10        | 0.874 | 0.8            | 0.1            | 0.1          | 5.0       | 71.0           |
| 11        | 0.667 | 1.0            | 0.1            | 0.001        | 5.0       | 53.0           |
| 12        | 0.923 | 1.0            | 0.9            | 0.1          | 5.0       | 61.0           |

Table 1. Bayesian optimization of LightGBM model.
Bagging fraction is normally used to speed training and/or reduce overfitting, since it resamples the data for a specific iteration using only the percentage specified in the parameter. Since LightGBM is a gradient boosting machine implementation, it is needed to define how often boosted trees will be created in a bootstrap. In this case, 5 will be used, which means that every five trees one would be created using a bootstrap containing between 80% to 100% of the data. This bagging fraction parameter, as can be seen in the Table 1, was optimized to 1, i.e., 100% of the data in each bootstrap. This behavior is expected since the Bayesian process was optimizing for F1-score and not for training time.

Feature fraction behaves similarly to the Max features parameter in Random Forest. Its goal is to reduce the number of predictors used in each iteration to improve training times and reduce the variance across all trees. As can be seen in the Table 1, feature fraction parameter has a behavior similar to Max features in Random Forest, tending to higher values, 0.7 in the case of Random Forest, 0.9 in this case. This usually means that most features constitute noise and offer no additional predictive value.

Following the previous parameters, comes the parameter that most influences the model’s performance, the maximum depth of the tree. LightGBM uses a leaf-wise (best-first) tree growth, i.e., it chooses to grow the branch whose split leads to maximum reduction of impurity. By doing so, LightGBM can often achieve better results than methods that use depth-wise growth. Nevertheless, the algorithm still provides a maximum depth parameter, which means that the leaf-wise growth is limited in height by this parameter, in order to reduce overfitting by controlling the variance of each tree. Looking at Table 1 it can be seen that the maximum depth can vary between 5 and 9, with the maximum value being chosen (9). Another common parameter is the number of leaves which, contrarily to maximum depth parameter, regulates the growth of the tree in a leaf-wise manner. However, the objective of this parameter is the same, it is used to control overfitting. The analysis of Table 1 suggests again that this parameter should be maximized using 80 as the maximum number of leaves. This points to the potential increase in the model score, by increasing the number of leaves. However, the computational costs of increment the number of leaves would increase an already long training time of 8 min. Finally, the last parameter is learning rate which, as in Rusboost, is used to delay the convergence of the model, since the slower the model learns the less likely it is to get stuck in local minima. This parameter is easier to optimize when there are many trees. Thus, in this case, with 350 trees, only lower learning rates can be explored. As such, Bayesian optimization chose the highest value with which the model could converge in only 350 trees - 0.1 (see Table 1).

5.5 Experimental Results and Analysis

To better understand the performance of the algorithms, a macro-averaged score of the recall, precision and F1-score was calculated for each model (see Fig. 13). With this information it is possible not only to gain insight into the performance of the algorithm but also into the trade-offs between precision and recall.
It is possible to notice that Random Forest, LightGBM and decision trees stand as the best performing models with 92%, 91% and 89% F1-score, respectively. This can be attributed, in the case of Random Forest and decision trees, to the depth of the trees used, since deepest trees achieve better results. In the case of LightGBM the trees depth is still high (9 levels), but the most important factor is the 80 leaf nodes that increase the method’s ability to learn complex behavior together with boosting strategies. On the other hand, the rest of the boosting algorithms (Adaboost and RusBoost), performed poorly, attaining only 25% and 56% F1-score, respectively. This could be explained, as previously noted, for two reasons: first the depth of the trees, since both algorithms were modeled using only trees of depth 3 to not incur in unnecessary variance; and the low number of trees that did not provide enough iterations for the model to converge. As such, the solutions to these problems would be to increase the number of trees from 150 to 600, for example, to try to regulate the learning rate in order to converge the solution; and use deeper trees, preferably not exceeding 10 in depth, with a risk of performance degradation. Another remarkable property is the recall of the balanced Random Forest algorithm, which is highest among all algorithms along with LightGBM, despite having an exceedingly low precision. This indicates that the undersampling of the majority class had significant impact on reducing the number of false negatives but it also increased the amount of benign traffic labeled as an attack (false positives). This trade-off could be better regulated by adjusting the probability thresholds of the balanced Random Forest.

6 Conclusion

In this paper we present SAFECARE Cyber Threat Monitoring System and detail the IT threat detection systems, by comparing the attack detection performance of six different ensemble algorithms (Adaboost, Rusboost, Random Forest, Balanced Random Forest, Decision Tree and LightGBM) using the CICIDS2017 dataset. For each algorithm a study of the optimal parameter was made using Grid Search and Bayesian search approaches. After, the parameter optimization three different metrics were chosen to compare the algorithms.
performance: recall, F1-score and precision. The results identified, using a 95% confidence interval, Random Forest, LightGBM and Decision Trees as the best algorithms with no significant difference in performance and the rest of the algorithms in the following order: Rusboost, Balanced Random Forest, Adaboost. The selected techniques are now deployed and being used for attack detection on data resultant from the simulation of different attacks under Airbus cyber range tool.

**Acknowledgements.** This work has received funding from European Union’s H2020 research and innovation programme under SAFECARE Project, grant agreement no. 787002.

**References**

1. Agusta, Z., Adiwijaya, A.: Modified balanced random forest for improving imbalanced data prediction. Int. J. Adv. Intell. Inform. 5(1), 58–65 (2019). https://doi.org/10.26555/ijain.v5i1.255. http://ijain.org/index.php/IJAIN/article/view/255
2. Alshamrani, A., Myneni, S., Chowdhary, A., Huang, D.: A survey on advanced persistent threats: techniques, solutions, challenges, and research opportunities. IEEE Commun. Surv. Tutor. 21(2), 1851–1877 (2019)
3. Aslan, A., Samet, R.: A comprehensive review on malware detection approaches. IEEE Access 8, 6249–6271 (2020)
4. Brown, G.: Ensemble Learning, pp. 393–402. Springer, Boston (2017). https://doi.org/10.1007/978-1-4899-7687-1_252
5. University of California, I.: KDD cup 1999 data, March 2018. http://mlexplained.com/2018/01/05/lightgbm-and-xgboost-explained/
6. for Cybersecurity, C.I.: Intrusion detection evaluation dataset (CICIDS 2017), March 2018. https://www.unb.ca/cic/datasets/ids-2017.html
7. Dewancker, I., McCourt, M., Clark, S., Hayes, P., Johnson, A., Ke, G.: A strategy for ranking optimization methods using multiple criteria. In: AutoML@ICML (2016)
8. Dhalwal, S.S., Nahid, A.A., Abbas, R.: Effective intrusion detection system using XGBoost. Information 9(7) (2018). https://doi.org/10.3390/info9070149
9. Feurer, M., Hutter, F.: Hyperparameter Optimization, pp. 3–33. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-05318-5_1
10. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference and Prediction, 2nd edn. Springer, New York (2009). https://doi.org/10.1007/978-0-387-84858-7
11. Hu, W., Hu, W., Maybank, S.: AdaBoost-based algorithm for network intrusion detection. IEEE Trans. Syst. Man Cybernet. Part B, Cybernet. 38, 577–83 (2008). https://doi.org/10.1109/TSMCB.2007.914695. A Publication of the IEEE Systems, Man, and Cybernetics Society
12. Koehrsen, W.: An implementation and explanation of the random forest in python, August 2018. https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76
13. Latah, M., Toker, L.: Towards an efficient anomaly-based intrusion detection for software-defined networks, March 2018
14. Mazini, M., Shirazi, B., Mahdavi, I.: Anomaly network-based intrusion detection system using a reliable hybrid artificial bee colony and AdaBoost algorithms. J. King Saud Univ. Comput. Inf. Sci. (2018). https://doi.org/10.1016/j.jksuci.2018.03.011
15. Parr, T., Turgutlu, K., Csiszar, C., Howard, J.: Beware default random forest importances, March 2018. https://explained.ai/rf-importance/
16. Polikar, R.: Ensemble based systems in decision making. IEEE Circuit Syst. Mag. 6, 21–45 (2006). https://doi.org/10.1109/MCAS.2006.1688199
17. Probst, P., Boulesteix, A.L.: To tune or not to tune the number of trees in random forest. J. Mach. Learn. Res. 18(1), 6673–6690 (2017). http://dl.acm.org/citation.cfm?id=3122009.3242038
18. Ramadhan, M., Sitanggang, I., Rizky Nasution, F., Ghifari, A.: Parameter tuning in random forest based on grid search method for gender classification based on voice frequency. DEStech Trans. Comput. Sci. Eng. (2017). https://doi.org/10.12783/dtcse/cece2017/14611
19. Shiravi, A., Shiravi, H., Tavallaee, M., Ghorbani, A.: Toward developing a systematic approach to generate benchmark datasets for intrusion detection. Comput. Secur. 31, 357–374 (2012). https://doi.org/10.1016/j.cose.2011.12.012
20. Utic, Z., Ramachandran, K.: Network attribute selection, classification and accuracy (NASCA) algorithm for intrusion detection systems, April 2017. https://doi.org/10.1109/THS.2017.7943463
21. W. Haines, J., P. Lippmann, R., J. Fried, D., Zissman, M., Tran, E.: 1999 DARPA intrusion detection evaluation: design and procedures, p. 188, February 2001
22. Yulianto, A., Sukarno, P., Suwastika, N.: Improving AdaBoost-based intrusion detection system (IDS) performance on CIC IDS 2017 dataset. J. Phys. Conf. Ser. 1192, 012018 (2019)
23. Zhu, J., Rosset, S., Zou, H., Hastie, T.: Multi-class AdaBoost. Stat. Interface 2 (2006). https://doi.org/10.4310/SII.2009.v2.n3.a8