Resource efficient boosting method for IoT security monitoring.

ZAKARIYYA, I., AL-KADRI, M.O. and KALUTARAGE, H.

2021

Resource Efficient Boosting Method for IoT Security Monitoring

Idris Zakariyya, M. Omar Al-Kadri, Harsha Kalutarage
School of Computing Science, Robert Gordon University
Aberdeen, United Kingdom
i.zakariyya@rgu.ac.uk

***Abstract*—Machine learning (ML) methods are widely proposed for security monitoring of Internet of Things (IoT). However, these methods can be computationally expensive for resource constraint IoT devices. This paper proposes an optimized resource efficient ML method that can detect various attacks on IoT devices. It utilizes Light Gradient Boosting Machine (LGBM). The performance of this approach was evaluated against four realistic IoT benchmark datasets. Experimental results show that the proposed method can effectively detect attacks on IoT devices with limited resources, and outperforms the state of the art techniques.**

***Index Terms*—Machine Learning, Internet of Things, Resource Constraint, Light Gradient Boosting Machine.**

I. INTRODUCTION

The Internet of Things (IoT) technology is advancing with the proliferation of physically connected objects. IoT integrated multiple devices into networks [1] to provide efficient and intelligent services at a minimum cost. Therefore, IoT is the driving force behind various advanced automation systems.

Despite their diverse implementation and adoption in different sectors, the security and privacy of these interconnected smart things pose a significant challenge. The need for robust security techniques in response to their resource limitation escalates. Based on the available resource, IoT networks have to provide effective security mechanisms that can monitor and detect severe cyber threats.

The adoption of Machine Learning (ML) techniques are gaining popularity as an approach with wider applications in many areas. In the context of IoT, the deployed anomaly detection system in [2] improved the detection of malicious activities on smart phone devices. The deeper approach in [3] can detect Distributed Denial of Servive (DDoS) botnet attacks on consumer IoT devices. The appealing nature of ML that dignifies its practical implementation in various fields is the capability of developing a model that can learn the statistical distribution of complex and higher dimensional datasets.

Due to the motivational application of ML methods on IoT networks, Decision Tree Ensemble Methods (DTEMs) are employed to mitigate various security threats. An overview of such implementation is presented in [4]. However, most of the underlying DTEMs are computationally expensive with complex and larger dataset [5]. Their practical implementation required intensive memory and time resources. This limits their direct deployment for IoT security monitoring.

Based on this observation, we develop an optimized boosting method for resource constraint IoT devices. It adopted an ML model and factored out its less computational expensive parameters. It can accurately discern attack and regular traffic on IoT networks. It provides a robust implementation in IoT security monitoring, while the best trade-off depends on the datasets and selected ML model.

The rest of this paper is organized as follows. Background and related studies are presented in Section II. The proposed approach is described in Section III. Section IV reports the evaluation process and Section V presents results. Section VI concludes the paper with a future research direction.

II. BACKGROUND AND RELATED WORK

In this section, we present a brief background on DTEMs and previous studies conducted related to this paper.

Ensemble procedure is a combination based technique that involves the strengthening of a multiple weaker classification model to produce a better model. The fine-tune classifier model is utilized to predict new data instances.

Bagging is an ensemble decision tree algorithm that manipulates the training data instances to improve classification model performance [6]. The process of generating another set of data with replacement from the original data is called *bootstrapp replicate* while the technique is referred to as *bootstrap aggregation* [7].

Boosting is an ensemble method that ensures the reduction of the classification error generated from the previous classifier. This technique is more sensitive to model over-fitting on noisy data with extensive training iteration [8].

Despite their computational expenses in the accumulation of multiple learners to make a decision, DTEMs offer an appealing advantage in various fields. Especially for solving a plethora of many ML challenges. From the literature, several comprehensive surveys on ensemble learners are available [9], [10] and [11]. In IoT networks, ensemble learners manifest the concept of incorporation to detect attacks captured from multiple devices. The integrated statistical [12] approach can identify irregular events from various IoT network traffics.

Motivated by its bagging and feature selection properties, the Random Forest (RF) method has extensive applications. The authors in [13] utilized it with real-time implementation. Also, Resende et al. in [14] outline its employment opportunities

in network security monitoring. Hassan et al. [15] adopted it for network intrusion detection. Empirically validated the approach using an extended version of the KDD 99 dataset. In that context, the technique is less computationally expensive than the Support Vector Machine (SVM) model. The evaluation results reported in [16], shows its effectiveness. It outperforms the SVM and Artificial Neural Network model as tested with smaller datasets. Also, Singh in [17] adopts and implements it for peer to peer real-time botnet detection. However, there is no consideration for memory consumption from the described [17] approach.

In contrast, an optimized DTEMs based on RF was proposed [18] and tested with a small scale Iris dataset. Such implementation over insufficient data records restricted the model adoption capability. This limits the approach applicability in various fields. Especially in the IoT resources constraint domain.

In terms of efficient performance on sparse data, a scalable boosted algorithm was proposed in [19]. Also, a lighter gradient boosting decision tree method based on feature sub sampling that require larger gradient was proposed in [20]. The central assumption made is based on the features dimension reduction to speed up the model classification process. However, this decision cannot be generalized within various datasets as an approach to improve model performance.

Series of ML optimization methods have been proposed for classification tasks with Deep Neural Networks (DNN) in [21] and [22]. The optimized loss function in [23] outperforms the popular cross-entropy technique for image classification. Further, a complex and robust deeper optimization approach has been successfully implemented in [24].

In particular, no existing work in the boosted decision tree literature that automatically optimizes learning parameters for LGBM in the context of IoT. In this paper, we demonstrate such a useable approach that reduces memory and time resources consumption. The method ensures the selection of relevant parameters suitable for IoT resource constraint environment and multidimensional scalable datasets.

III. RESOURCE EFFICIENT BOOSTING METHOD

The process of finding an optimized decision tree classification model can be considered as a challenging task. This is due to the need for intensive parameters tuning in achieving the state of the art performance. For example, optimization of the Gradient Boosting Method (GBM) requires attentive parameter tuning to build a base learner that maximally correlated with the negative gradient of a loss function [25].

A. Light Gradient Boosting Machine

Light Gradient Boosting Machine (LGBM) is a decision tree algorithm based on gradient sampling of data instances with smaller gradient and exclusive feature bundling [20]. This technique has proven successful with a lesser iteration on the training data instances. However, practical implementations of LGBM need various parameter tuning. This is challenging with multidimensional and scalable datasets. Unfortunately, it is difficult to find these optimum parameters that are memory and time resources-efficient with larger training data samples. An attentive configuration is needed in setting the relevant boosted learning architectures for the task of regression, binary, and multiclass classification.

B. LGBM Hyperparameters

LGBM learning model has various hyperparameters to consider for optimization phases and implementation, particularly for the task of binary classification. These include the number of leaves, feature fraction, bagging fraction, bagging frequency, learning rate, and a regularization term. Table I described the range of values of these hyperparameters. There is a recommendation in varying each scale, as described in the LGBM module [26]. Regularization alpha as a constraint can be greater than 0.0. Feature and bagging fraction must be set within [0, 1]. Enabling bagging with its frequency set to non zero value can facilitate efficient model learning.

For the task of efficient resource utilization, the learning rate, and the regularization term are selected in the range of [0.0001, 0.1]. The constraint bagging and feature fraction are utilized within [0, 1]. The constraint number of leaves may cause model over-fitting with increases in computational cost, minimum scales of 2 are initialized and incremented sequentially. Proper configuration of these hyperparameters can minimize time and memory resources consumption.

Primarily, the grid search technique is employed to select the best parameters configuration among various learning models. However, this approach focuses on providing better prediction performance. Yet, there is a challenge in stabilizing the threshold between memory usage and accurate prediction.

Algorithm 1 Proposed method

Dataset D = $\{(x_1, y_1), (x_2, y_2),..., (x_n, y_n)\}$
Normalized $D_n = \{D_j\}$, $j = 1 \dots m$
Train data $X \subseteq D_n$
Test data $X' \subseteq D_n$ ▷ $X \cup X' = D_n$, $X \cap X' = \emptyset$
M = Classifier model
M_e = efficient classifier model
$\{Z\}$ = Parameters and hyperparameters
$L = |Z|$
$\{Z_o\}$ optimized Z
for $k = 1$ to L **do**
    EFFICIENT($Z[\,]$) ▷ Call to EFFICIENT in Alg. 2
    $M_e = M(X, Z_e)$ ▷ M_e depend on Z_e and X
end for
Predictive $E = M_e(X')$ ▷ Prediction using M_e and X'

An optimized method for this purpose have been proposed and presented in Algorithm 1. During training iteration, the described algorithm can select the optimum parameters of the LGBM model. Based on minimal fitted memory and running times. The selection procedure is in Algorithm 2. As demonstrated, the function efficient parameter, initialize the memory and time consumption of the classifier model. Then iterate sequentially to the remaining parameters to return an

TABLE I: LGBM hyperparameters.

Hyperparameter	Minimum	Maximum
Bagging Fraction	0.0	1
Feature Fraction	0.0	1
Number of Leaves	1	131072
Learning Rate	0.0	0.1
Regularizer	0.0	0.1

efficient model. This model is employed to predict a testing dataset with better accuracy.

Algorithm 2 Resource efficient

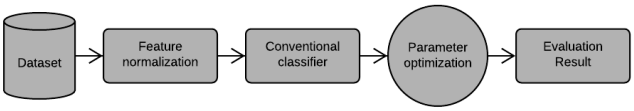
Training samples T = $\{(x_i, y_i)\}$, $i = 1 \dots n$
Testing samples X= $\{(x_i, y_i)\}$, $i = 1 \dots m$
Classifier C
C model parameters P = $\{p_j\}$, $j = 1 \dots l$
m_f, t_f fitted memory and fitted run times of C
a_{p_j} prediction accuracy of $C(p_j)$
function EFFICIENT($P[\,]$)
    $l \leftarrow length(P)$
    $t_{fp_1} \leftarrow t_{C(p_1,T)}$
    $m_{fp_1} \leftarrow m_{C(p_1,T)}$
    $min(t) \leftarrow t_{fp_1}$
    $min(m) \leftarrow m_{fp_1}$
    $a_{p_1} \leftarrow Cp_1$(X)
    $p_t \leftarrow a_{p_1}$
    for $k \leftarrow 2$, l **do**
        $t_{fp_j} \leftarrow t_{C(p_j,T)}$
        $m_{fp_j} \leftarrow m_{C(p_j,T)}$
        $a_{fp_j} \leftarrow Cp_j$(X)
        while $((a_{p_j} \geq p_t) \vee (a_{p_j} \leq p_t))$ **do**
            if $((t_{fp_j} < min(t)) \wedge (m_{fp_j} < min(m)))$ **then**
                $min(t) \leftarrow t_{fp_j}$
                $min(m) \leftarrow m_{fp_j}$
            end if
            break
        end while
    end for
    return $C(p_j, min(t_{fp_j}, m_{fp_j})$
end function
Validation:
$F \leftarrow C$(X)

C. Parameters Optimization Procedure

Optimization of a decision tree classification model is a requirement to maintain generality on resource-intensive tasks. Particularly for their deployment to resources constraint IoT devices. Careful employment of optimum parameters combination can enhance model performance with minimum resources consumption.

The visualization in Figure 1, is the process diagram for the proposed method. It required datasets and a conventional ML model alongside with the parameters that occur in training. It can accept data, normalize it, and output a model with optimized parameters. It employed dataset produce from the collection of realistic benign and malicious traffics using simulated IoT devices. For the task of binary classification, the efficient model can accurately differentiate regular traffic from attack traffic.

Fig. 1: Resource efficient process diagram



The pseudocode of this implementation procedure is presented in Algorithm 2. The foremost steps in Algorithm 2, is the traditional classifier model definition with its parameters composition. During training, parameters are iterated sequentially to find those that efficiently fit a model. The return parameters are those that are effective in terms of resource utilization. Testing data samples are validated using the updated model.

D. Implementation

In practice, the employment of the parameter grid [27] is due to the various series of training sessions that occur for the discovery of optimum parameters with fewer resources. The parameter grid [27] can store multiple parameters. Training and testing was implemented on Spyder [28] version 3.3.3 stable python IDE. Memory and iteration time has been profiled using the integrated *psutil (process and systtem utilities)* and *time* python modules. Computation were conducted on a personal machine that has intel Xeon E5-2695(4 core) CPUs running at 2.10 GHz with 16.0 GB installed memory. Code for the approach implementation is open-sourced at: https://github.com/izakariyya/Resource_Constraint_Algorithm

IV. EVALUATION

In this section, we present the experimental evaluation of the proposed method with the description of the benchmark datasets selected for IoT security monitoring.

A. Datasets and Preprocessing

The evaluation experiments used four accessible IoT datasets, N-BaIoT [29], Bot-IoT [30], Bot-10 [30], and Unsw [31]. Each dataset consists of various attacks along with normal traffics activities. Particularly, Bot-IoT with multiple categorizations of different botnet attacks. The dataset composed of 72 million records with 16.7 GB of CSV format file, while 10% of the data is made publicly available [30] for model evaluation.

The choice of these datasets allowed frequent model training with thorough experimentation. Each tested dataset are categorized into 70% training and 30% testing records. The datasets are described briefly in Table II. All data records are normalized using the employed min-max standard normalization formula described in Equation 1. The notation X in Equation1, represents the value of vector X, while X_{max} and X_{min}

TABLE II: Datasets.

Dataset	Training	Tests	Feature dimensions
N-BaIoT	509865	218514	115
Unsw	115264	49400	43
Bot-IoT	467965	200557	34
Bot-10	1247596	534684	10

TABLE III: Initial and optimum hyperparameters.

Hyperparameter	Initial	Optimum
Feature Fraction	0.1	0.4
Bagging Fraction	0.1	0.4
Bagging Frequency	2	2
Number of Leaves	31	2
Learning Rate	0.1	0.0001
Regularizer	0.0	0.0001

represents the maximum and minimum values of the vector X. These normalized datasets are within the range of [0,1].

$$Normalized(X) = \frac{X - X_{min}}{X_{max} - X_{min}} \quad (1)$$

B. Optimized LGBM Hyperparameters

The most relevant hyperparameters discovered using the proposed method are described in Table III. These include the initial values for the unoptimized LGBM model and optimum values returned by the efficient boosted algorithm.

V. RESULTS

This section presents the analysis and discussion of the experimental results from the implementation of the proposed boosting method. Memory and time usage of the optimization phase are compared with the tested techniques.

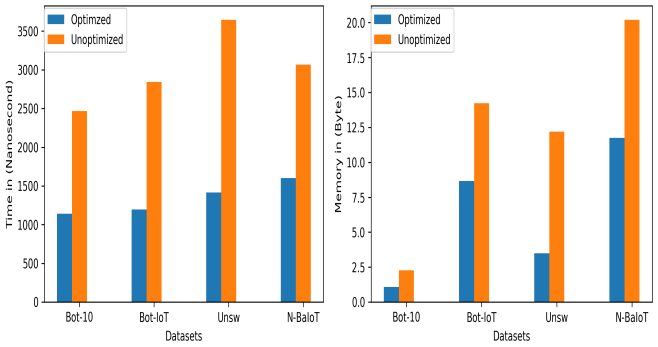
A. Testing Speed

The visualization in Figure 2(a) is the testing times needed to evaluate the proposed method against each datasets record. It is efficiently faster than the unoptimized model. It demonstrates reduced classification time in processing each sample of the tested data.

The proposed method is capable of saving processing time resources across datasets. As compared with the conventional model in Figure 2(a), it saved 53.79%, 57.89%, 61.11% and, 47.76% of times for validating a sample of Bot-10, Bot-IoT, Unsw and, N-BaIoT, respectively.

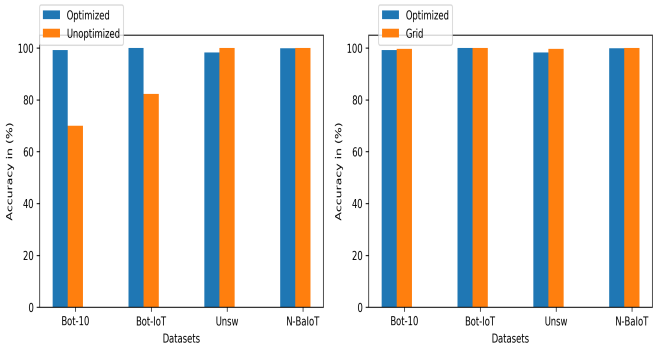
B. Testing Memory

The memory unit consumption comparison in testing each data record with the proposed approach is presented in Figure 2(b). It requires lesser memory. It saved 52.69%, 39.17%, 71.43% and, 41.78% of test memory for each record of Bot-10, Bot-IoT, Unsw, and, N-BaIoT, datasets, respectively. These results indicate its robustness and lightweight security monitoring advantages for IoT devices with overall improvement across benchmarks datasets. It suggests that real-time IoT security monitoring with the proposed approach can be beneficial.



(a) (b)

Fig. 2: Sample testing resource consumption (a) time and (b) memory.



(a) (b)

Fig. 3: Optimized testing accuracy comparison with (a) unoptimized LGBM and (b) grid search.

C. Testing Accuracy

The description in Figure 3(a) represents the testing accuracy that the proposed method provides for each dataset. Despite its resource reduction advantages, it also outperformed the unoptimized LGBM method in predicting the Bot-10 and Bot-IoT datasets accurately. The loss of accuracy by the unoptimized approach was due to the utilization of the default configuration parameters. These results indicate the capability of the optimized technique in detecting IoT attack traffic effectively.

The description in Figure 3(b) represents the testing accuracy comparison of the proposed method and the grid search technique. Despite its less computationally expensive, the prediction accuracy across each dataset is closer to that of the grid method.

D. Computational Performance Analysis

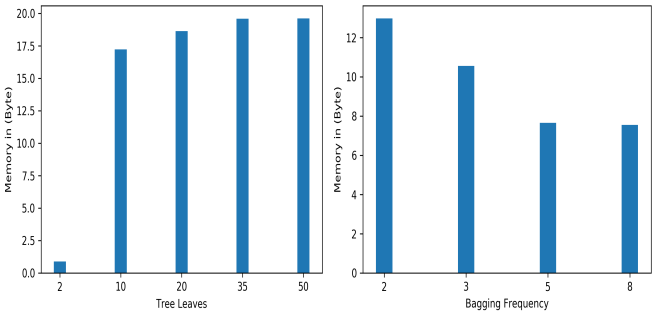
The reports in Table IV is the performance comparison for the tested datasets of the grid search and the proposed method. Regarding the resource consumption of each data record, the proposed approach is better. It required minimal memory and

TABLE IV: Grid and optimized method performance evaluation.

Dataset	Algorithm	Test memory (Byte)	Test time (Nanosecond)
N-BaIoT	Grid	15.88	2059.36
	Optmize	11.75	1601.73
Unsw	Grid	5.72	1619.43
	Optmize	3.482	1417.00
Bot-IoT	Grid	11.19	1495.83
	Optmize	8.66	1196.67
Bot-10	Grid	1.52	1458.81
	Optmize	1.08	1140.86

TABLE V: Performance evaluation comparison on N-BaIoT Dataset.

Algorithm	Accuracy (%)	Test time (Nanosecond)	Test memory (Byte)
Random Forest	89.35	12813.82	3873.01
LogitBoost	89.14	13042.64	3874.00
SGB	89.53	13454.52	3875.00
AdaBoost	89.31	11440.91	3866.00
Optmize	99.90	1601.73	11.75



(a) (b)

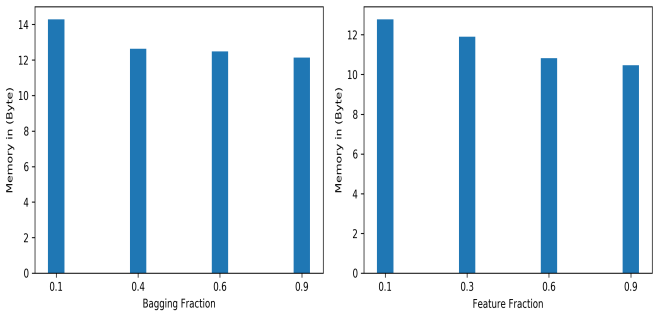
Fig. 4: Effect of (a) number of tree leaves and (b) bagging frequency on memory.

maintained reduced classification time. These results indicate its effectiveness and efficiency advantage.

The description in Table V is the computational performance of the employed algorithms against the N-BaIoT dataset sample. The proposed method indicate reductions in memory and time resources. It demonstrates a robust classification of attacks and regular traffic with an accuracy of 99.90%.

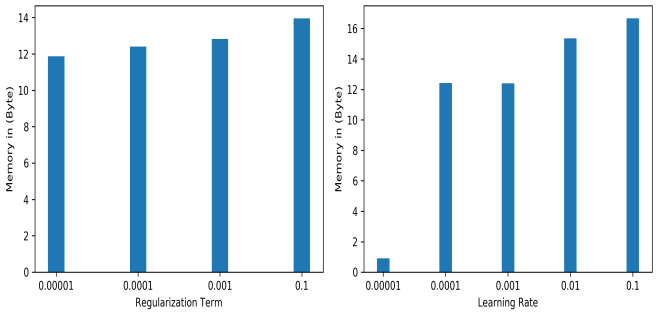
The demonstration in Figure 4 is the relationship between hyperparameters tuning against memory consumption. Figure 4a, indicates the effects of varying the constraint number of leaves on memory usage. The graph suggests smaller tree leaves values for lesser memory consumption. Also, Figure 4(b) indicates the memory consumed while altering the bagging frequency parameter.

The description in 5(a) is the sample memory consumption against the constraint bagging fraction. It suggests the selection of accurate value where resources are limited. Also, Figure 4(b),



(a) (b)

Fig. 5: Effect of (a) bagging fraction and (b) feature fraction on memory.



(a) (b)

Fig. 6: Effect of (a) regularizer and (b) learning rate on memory.

demonstrates how feature fraction facilitates memory resource consumption.

The illustration in Figure 6(a)is the sample memory consumption against the regualization term. Also, Figure 6(b) shows the impacts of learning rate on memory. It demonstrated that smaller values of these hyperparameters facilitate memory saving. This is useful in controlling the deployment of resource-hungry boosting algorithms.

VI. CONCLUSION AND FUTURE WORK

The increasing number and complexity of IoT devices motivate the development of a robust, efficient, and feasible security protection system. We present such an approach that utilized LGBM with optimized hyperparameters to lower computational cost for resource constraint IoT devices. Mainly due to the assumption that most traditional ML methods are computationally expensive for IoT security monitoring. The proposed technique is efficient and useable for IoT resources consumption reduction. It outperforms the conventional LGBM model tested with the initialized hyperparameters and the grid search technique. It is better than the five employed boosting algorithms for effective attack detection and lesser resource consumption. Regarding the SGB algorithm, it reduced the processing time and memory consumed during testing by 88.09% and 99.70% for each sample. These results motivate

follow-up research to enhance the method for real-time IoT security monitoring.

An essential step to validate the method externally would be to replicate this study results with regular and attack traffic captured from different real IoT devices. These include extensive network traffics captures during various IoT attacks generation.

Consideration of various datasets would allow the feasibility measurement of the proposed technique based on the amount and diversity of IoT traffic. We are more concerned about the behavior and variation of different IoT devices. At large, we want to investigate whether specific devices are more vulnerable than others. Further, we would like to explore more challenging ML techniques with other complex parameters and hyperparameters available in the literature.

VII. ACKNOWLEDGEMENT

We thank the School of Computing, Robert Gordon University for their assistance. This work was supported by the Petroleum Technology Development Fund (PTDF), Nigeria.

REFERENCES

[1] C. Cecchinel, M. Jimenez, S. Mosser, and M. Riveill, "An architecture to support the collection of big data in the internet of things," in *2014 IEEE World Congress on Services*. IEEE, 2014, pp. 442–449.

[2] P. Havinga, "Roads: A road pavement monitoring system for anomaly detection using smart phones," in *Big Data Analytics in the Social and Ubiquitous Context: 5th International Workshop on Modeling Social Media, MSM 2014, 5th International Workshop on Mining Ubiquitous and Social Environments, MUSE 2014, and First International Workshop on Machine Learning for Urban Sensor Data, SenseML 2014, Revised Selected Papers*, vol. 9546. Springer, 2016, p. 128.

[3] G. Han, L. Xiao, and H. V. Poor, "Two-dimensional anti-jamming communication based on deep reinforcement learning," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2017, pp. 2087–2091.

[4] N. Moustafa, J. Hu, and J. Slay, "A holistic review of network anomaly detection systems: A comprehensive survey," *Journal of Network and Computer Applications*, vol. 128, pp. 33–55, 2019.

[5] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*, 2014, pp. 583–598.

[6] T. Jiang, J. Li, Y. Zheng, and C. Sun, "Improved bagging algorithm for pattern recognition in uhf signals of partial discharges," *Energies*, vol. 4, no. 7, pp. 1087–1101, 2011.

[7] C. Bergmeir, R. J. Hyndman, and J. M. Benítez, "Bagging exponential smoothing methods using stl decomposition and box–cox transformation," *International journal of forecasting*, vol. 32, no. 2, pp. 303–312, 2016.

[8] C. Cortes, M. Mohri, and U. Syed, "Deep boosting," 2014.

[9] H. M. Gomes, J. P. Barddal, F. Enembreck, and A. Bifet, "A survey on ensemble learning for data stream classification," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–36, 2017.

[10] L. Rokach, "Decision forest: Twenty years of research," *Information Fusion*, vol. 27, pp. 111–125, 2016.

[11] Y. Ren, C. Domeniconi, G. Zhang, and G. Yu, "Weighted-object ensemble clustering: methods and analysis," *Knowledge and Information Systems*, vol. 51, no. 2, pp. 661–689, 2017.

[12] N. Moustafa, B. Turnbull, and K.-K. R. Choo, "An ensemble intrusion detection technique based on proposed statistical flow features for protecting network traffic of internet of things," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4815–4830, 2018.

[13] J. Browne, D. Mhembere, T. M. Tomita, J. T. Vogelstein, and R. Burns, "Forest packing: Fast parallel, decision forests," in *Proceedings of the 2019 SIAM International Conference on Data Mining*. SIAM, 2019, pp. 46–54.

[14] P. A. A. Resende and A. C. Drummond, "A survey of random forest based methods for intrusion detection systems," *ACM Computing Surveys (CSUR)*, vol. 51, no. 3, pp. 1–36, 2018.

[15] M. A. M. Hasan, M. Nasser, B. Pal, and S. Ahmad, "Support vector machine and random forest modeling for intrusion detection system (ids)," *Journal of Intelligent Learning Systems and Applications*, vol. 2014, 2014.

[16] T. Han, D. Jiang, Q. Zhao, L. Wang, and K. Yin, "Comparison of random forest, artificial neural networks and support vector machine for intelligent diagnosis of rotating machinery," *Transactions of the Institute of Measurement and Control*, vol. 40, no. 8, pp. 2681–2693, 2018.

[17] K. Singh, S. C. Guntuku, A. Thakur, and C. Hota, "Big data analytics framework for peer-to-peer botnet detection using random forests," *Information Sciences*, vol. 278, pp. 488–497, 2014.

[18] Y. Mishina, R. Murata, Y. Yamauchi, T. Yamashita, and H. Fujiyoshi, "Boosted random forest," *IEICE TRANSACTIONS on Information and Systems*, vol. 98, no. 9, pp. 1630–1636, 2015.

[19] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[20] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in neural information processing systems*, 2017, pp. 3146–3154.

[21] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," *arXiv preprint arXiv:1702.05659*, 2017.

[22] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *Siam Review*, vol. 60, no. 2, pp. 223–311, 2018.

[23] S. Gonzalez and R. Miikkulainen, "Improved training speed, accuracy, and data utilization through loss function optimization," *arXiv preprint arXiv:1905.11528*, 2019.

[24] J. T. Barron, "A general and adaptive robust loss function," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4331–4339.

[25] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Frontiers in neurorobotics*, vol. 7, p. 21, 2013.

[26] "LGBM tuning parameters," https://https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html, accessed: 2020-01-05.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[28] P. Raybaut, "Spyder: Scientific python development environment, 2009–," *URL" https://github. com/spyder-ide/spyder".[Online*, 2017.

[29] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici, "N-baiot—network-based detection of iot botnet attacks using deep autoencoders," *IEEE Pervasive Computing*, vol. 17, no. 3, pp. 12–22, 2018.

[30] N. Koroniotis, N. Moustafa, E. Sitnikova, and B. Turnbull, "Towards the development of realistic botnet dataset in the internet of things for network forensic analytics: Bot-iot dataset," *Future Generation Computer Systems*, vol. 100, pp. 779–796, 2019.

[31] N. Moustafa and J. Slay, "Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set)," in *2015 military communications and information systems conference (MilCIS)*. IEEE, 2015, pp. 1–6.