Complex & Intelligent Systems (2023) 9:3787–3799
https://doi.org/10.1007/s40747-022-00809-3

ORIGINAL ARTICLE

Multi-layer stacking ensemble learners for low footprint network intrusion detection

Saeed Shafieian[1] · Mohammad Zulkernine[1]

Received: 15 July 2021 / Accepted: 16 June 2022 / Published online: 5 July 2022

Abstract
Machine learning has become the standard solution to problems in many areas, such as image recognition, natural language processing, and spam detection. In the area of network intrusion detection, machine learning techniques have also been successfully used to detect anomalies in network traffic. However, there is less tolerance in the network intrusion detection domain in terms of errors, especially false positives. In this paper, we define strict acceptance criteria, and show that only very few ensemble learning classifiers are able to meet them in detecting low footprint network intrusions. We compare bagging, boosting, and stacking techniques, and show how methods such as multi-layer stacking can outperform other ensemble techniques and non-ensemble models in detecting such intrusions. We show how different variations on a stacking ensemble model can play a significant role on the classification performance. Malicious examples in our dataset are from the network intrusions that exfiltrate data from a target machine. The benign examples are captured by network taps in geographically different locations on a big corporate network. Among hundreds of ensemble models based on seven different base learners, only three multi-layer stacking models meet the strict acceptance criteria, and achieve an F1 score of 0.99, and a false-positive rate of 0.001. Furthermore, we show that our ensemble models outperform different deep neural network models in classifying low footprint network intrusions.

Keywords Network intrusion detection · Anomaly detection · Ensemble learning · Stacking ensemble learning · Low footprint intrusion

Introduction

A network intrusion is any attack on, or misuse of network resources. There are two major network intrusion detection techniques; misuse detection and anomaly detection [1]. Misuse detection approaches are usually signature-based methods that look for specific patterns, such as byte or packet sequences in the network traffic that are already known to be malicious. Therefore, they basically define the malicious traffic patterns, and any traffic that matches those will be flagged.

Anomaly-based intrusion detection systems, however, mainly employ machine learning techniques to detect network intrusions. These techniques train models on both benign and malicious network traffic to build a model that would be able to classify new, unseen network data as malicious or benign.

In this paper, we focus on Ensemble learning methods for low footprint network intrusion detection. Ensemble learning methods combine base classifiers (learners) in different ways using techniques such as bagging, boosting, or stacking. The classification performance can be usually boosted by combining many weak classifiers of the same type, but each with different hyperparameters, or by training the same type of learners on different datasets. Ensemble classifiers can also be built by combing a few models each built using a different strong classifier.

Ensemble learning aims to minimize two sets of errors; variance and bias. Variance is error from sensitivity to small fluctuations in training set; high variance can cause overfitting. Bias is erroneous assumptions in the model; high bias can cause underfitting. Ensemble methods are able to reduce variance if the training sets are completely independent and predictions are averaged. Averaging models in an ensemble

✉ Saeed Shafieian
saeed@cs.queensu.ca

Mohammad Zulkernine
mz@queensu.ca

1 School of Computing, Queen's University, Kingston, Canada

can also reduce bias by fitting one component at a time using boosting techniques. Variance and bias usually vary in different directions, and as a result, finding a balance between the two is very important.

Bootstrap aggregation, or bagging in short, creates an ensemble model by training same base classifiers on random subsets of training data called *bootstrap samples*. Each bootstrap sample is formed by random sampling the training set with replacement. This process is called *bootstrapping*, and creates training sets with overlapping examples. The predictions are then combined (aggregated) to produce the final result. Aggregation is done by normally majority voting, or averaging the probabilities for each predicted class. Bagging reduces variance without affecting bias. Base learners are trained independently in bagging; therefore, the training process can be done in parallel. One of the most well-known and mostly used bagging classifier is Random Forest, which combines decision trees [2]. Trees in a Random Forest can be shallow or deep. Shallow trees have less variance, but higher bias. Deep trees, on the other hand, have low bias but high variance.

In boosting, only one classification algorithm is used, for example a decision tree. The learning is done in a sequential order. Each classifier tries to boost the previous model by giving more weights to the misclassified examples in the training set. Boosting is a strong alternative to bagging. Instead of aggregating predictions, boosters turn weak learners into strong learners by focusing on where the individual models, usually Decision Trees, did not perform well. Since the training is done in a sequential and iterative manner, it cannot usually be done in parallel. One of the most well-known and used boosting methods is eXtreme Gradient Boosting (XGBoost) [3].

Stacking is different from bagging and boosting in a number of ways. First, the base learners are normally heterogeneous in stacking, meaning that they each use a different classification algorithm. This is different from bagging and boosting that usually use homogeneous learners. The second difference is in how the base classifiers are combined. In bagging and boosting, the base classifiers are combined using a deterministic algorithm, such as majority voting or averaging. However, a *meta* classifier such as *Logistic Regression* is trained in stacking to combine the base models.

In a two-layer stacking, all the base classifiers perform their predictions, and then, a *meta* classifier is used to combine these results into a final prediction. Figure 1 shows how the simple two-layer stacking works. Usually, Logistic Regression or a neural network is usually trained as the meta classifier to combine the base models.

There are several different parameters that need to be considered in creating a stacking ensemble model. These parameters include:

- The base classifiers on each layer
- Each base classifier's hyperparameters
- The meta classifier
- The meta classifier's hyperparameters.

In machine learning, a *hyperparameter* is a parameter that is set before model training begins, and controls the learning process in some way. For example, the "number of trees" in Random Forest is a hyperparameter. Hyperparameter optimization, also known as hyperparameter tuning, is the process of finding the most optimal hyperparameters for a learning algorithm. For instance, the number of trees and the max depth of each tree can be optimized in Random Forest, so that a better classification is resulted. Hyperparameter optimization techniques mostly use one of the optimization algorithms, such as Grid Search, Random Search, or Bayesian Optimization. We discuss how variations of the above parameters can affect the performance of the stacking ensemble classifiers.

One of the biggest challenges in using machine learning to detect network intrusions is having access to real-world network traffic. The network traffic in large corporations is fundamentally different from those in small companies, or educational institutes. Usually, a model created based on data from a large corporate network can also be applied to smaller networks, but not vice versa [1].

In the network intrusion detection domain, one of the most popular datasets used by researchers is KDD'99 [4]. However, this dataset is too old now to be used as a benchmark for modern network intrusions. As a result, many researchers have stopped using this dataset, and some have started using the newer NSL-KDD dataset [5]. The NSL-KDD dataset is basically an improved version of the original KDD'99 dataset. This newer dataset excludes many duplicate records from the original dataset that could cause biases in classifiers. Nevertheless, none of these datasets suit the needs of today's intrusion detection. We use corporate network traffic that was captured in geographically different locations as our benign dataset. We generate malicious traffic from the AWS cloud targeting machines in a corporate network.

There has been a growing interest in both academia and industry to use Deep Learning in different domains. It has become the standard solution to problems in domains, such as natural language processing and computer vision. Nevertheless, deep learning has its own drawbacks, which would make it challenging to be used in some other domains including intrusion detection. These drawbacks include, but are not limited to the following [6]:

- Many hyperparameters to tune
- Theoretically infinite number of architectures to choose from

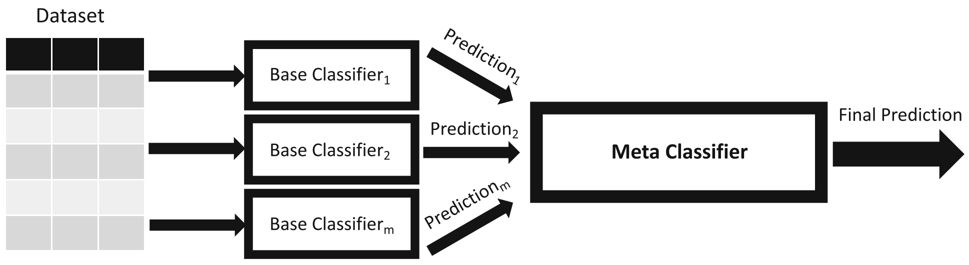

Fig. 1 Stacking technique in ensemble learning. A simple stacking ensemble model with base learners on the first layer, and the meta learner on the second layer. Meta learner combines the predictions of the base learners into a final prediction

- Opaqueness into results (black-box models)
- Large datasets normally required to obtain accurate results
- Relatively slower convergence on smaller datasets compared to some traditional machine learning algorithms.

In a simple neural network model such as multi-layer perceptron (MLP), the number of hidden layers and neurons on each layer creates many different network architectures to choose from. The choices of activation function for hidden layers, the solver for weight optimization, alpha for penalty, and learning rate for weight updates make many different hyperparameter sets to tune.

Deep neural networks suffer from another disadvantage, which is opacity into results. Most of the time, the models are black boxes due to the many layers, neurons, and weights. This renders the model as being *unexplainable*, and it is a major drawback in the intrusion detection domain. In operational real-world intrusion detection systems, having transparency into results is a key factor [1].

Another important drawback of deep learning is that a large training dataset is usually required to achieve high accuracy. This is practically a significant problem in some domains where training examples are difficult or costly to obtain. In domains such as intrusion detection, there are not as many malicious as benign network traffic examples. The other drawback of deep learning is longer running times for both training and test, which is mostly due to model complexity.

In this paper, we analyze and evaluate different ensemble learning methods as alternatives to deep learning and traditional machine learning models. We focus on detecting low footprint network intrusions. These are low-volume stealthy intrusions that do not transfer volumetric data *to* or *from* the target. Instances of such intrusions include Slow-Read distributed denial of service and data exfiltration using DNS tunneling.

We define strict acceptance criteria for classification performance, and perform experiments in detecting low footprint network intrusions. We show that among hundreds of different models, only three stacking ensemble models are able to meet the acceptance criteria. All these top-performing models use basic classification algorithms such as naive Bayes and decision tree as their base learners. We also show how small changes to each stacking ensemble model can have drastic impacts on the performance of the resulting model. We implement our experiments using scikit-learn [7], which is a widely used machine learning library in Python.

The rest of the paper is organized as follows. The section "Related work" presents related work. In the section "Stacking ensemble learning", we discuss stacking ensemble learning techniques. The section "Training and test datasets" presents our dataset and the feature selection process. In the section "Experimental results", we discuss our experimental results, and finally, the section "Conclusion" section concludes our work.

Related work

Ensemble learning methods have been used to detect network intrusions [8–12]. Shafieian et al. use Random Forest to detect slow-read distributed denial of service attacks [13]. They show how tuning the hyperparameters affects the classification performance. These authors also use heterogeneous ensemble models to detect the stealthy DNS tunneling intrusion [14]. They discuss how a weighted ensemble of Random Forest, k-NN, and MLP with different combination rules can be used to achieve the lowest false positive rates. Gao et al. propose a new ensemble model by adjusting the proportion of training data and setting up multiple decision trees [15]. They use the NSL-KDD dataset to test the accuracy of their proposed method. They propose an intrusion detection framework based on Support Vector Machine (SVM) ensemble with feature augmentation. Hsu et al. present a stacking ensemble learning model that consists of Autoencoder, SVM, and Random Forest models [16]. They use NSL-KDD, UNSW-NB15 [17], and a campus network log as their datasets. They compare their method to a number of different machine learning techniques. They find a weighted combination of the base learners through grid search. Zhong et al. propose an anomaly detection framework that is based on combining Autoencoder with long short-term memory (LSTM) network [18]. They use Damped Incremental Statistics algorithm to extract features from network traffic, and a weighted method to get the final abnormal score. They use

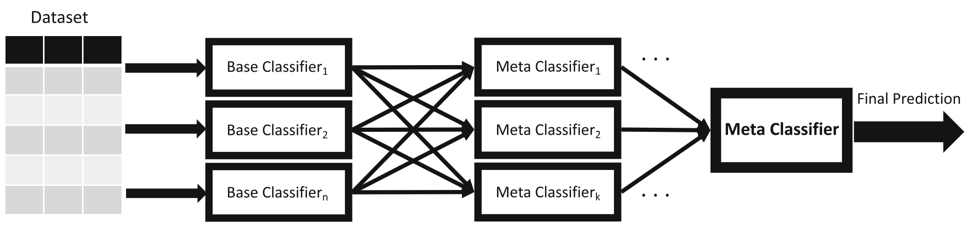

Fig. 2 Multi-layer stacking ensemble learning. First layer contains the base learners each producing a prediction on the dataset. Then, each prediction is fed to each meta learner on the second layer. Finally, a meta learner on the last layer combines the predictions of the previous layer into the final prediction

MAWILab [19] and IDS 2017 [20] datasets in their experiments. Mirsky et al. present a network intrusion detection system that uses an ensemble of Autoencoders [21]. They utilize a feature extraction framework to track the patterns of network channels. Their dataset consists of packets captured on a video surveillance network, where they send malicious traffic using available tools. Tama et al. propose a two-stage ensemble classifier that utilizes rotation forest and bagging [22]. They select features based on the classification performance of a reduced error pruning tree classifier on NSL-KDD and UNSW-NB15 datasets. Mirza et al. propose an ensemble of neural network, logistic regression, and decision tree using weighted majority voting [23]. They also use KDD'99 as their dataset for network intrusions.

Our work is different from existing work in several aspects. First, we create our own datasets as opposed to using old ones such as KDD'99 or NSL-KDD. Using these datasets makes it easy to compare the results with those of other researchers. Nonetheless, models and techniques developed using these outdated datasets may not be applicable to modern network intrusions [1]. Second, we use stacking ensemble models built using non-deterministic meta classifiers as opposed to deterministic techniques. More importantly, we define strict performance criteria which we show are only met by a few specific stacking ensemble learners. Our goal is to show that simple base learners can be combined in such a way that very high performing classifiers are resulted. This is in contrast with deep learning and other complex models that have a "black box" architecture.

Stacking ensemble learning

Similarly to bagging and boosting techniques, stacking methods combine base learners to create an ensemble model. However, there are two major differences between stacking and bagging or boosting. First, stacking usually uses heterogeneous base learners (different classification algorithms) as opposed to bagging and boosting that normally use homogeneous learners (same classification algorithms). Second, in bagging and boosting base learners are combined in a deterministic way. This is usually done using voting or a weighted sum. However in stacking, the base learners are combined using a non-deterministic algorithm via a *meta* learner.

As an example of stacking, one can use k-NN, MLP, and SVM as the base, and then employ logistic regression as the meta learner to combine the predictions. Here, k-NN, MLP, and SVM are the base learners, and Logistic Regression that combines their predictions is the meta learner. This means that logistic regression will learn how to combine the three predictions from the base learners, so that the best final classification is resulted. This process is done during the training process similar to other supervised learning methods.

Stacking can be done in two or multiple layers. In a two-layer stacking, all the base classifiers perform their predictions, and then, a *meta* classifier is used to combine the results into a final prediction. In the multi-layer stacking model, the predictions of each layer are fed to the next layer as the input. The last layer consists of a single meta classifier that combines the predictions into one final result. Figure 2 shows the multi-layer stacking technique. As the figure shows, there are n base learners on the first layer that are fully connected to the k meta learners on the second layer. The base learners are trained on the dataset, and then, each prediction from these base learners is fed to each meta learner on the second layer. The second layer meta learners are then trained on these predictions. There can be other layers of meta learners in this architecture. Finally, a meta learner on the last layer is trained to combine the predictions of its previous layer into the final prediction.

From a theoretical standpoint, any classification algorithm can be used on any of the layers in a stacking ensemble model. For instance, in the previous example, we could have used Logistic Regression as a base classifier, and SVM as the meta learner. Nonetheless, Logistic Regression or a neural network is usually a good choice for the final meta classifier. The flexibility in terms of the number of layers, the number of learners on each layer, and the type of each learner makes stacking very versatile and powerful for classification problems.

Training and test datasets

We use low footprint network intrusions in our datasets to train and test the classification models. These are data exfiltration intrusions that steal sensitive data from a victim's machine through tunneling [14]. We employ different tools to generate the malicious traffic to have a variety of malicious packets in our datasets. We generate the intrusions from virtual machines on Amazon AWS, targeting a machine in our lab.

We use Dnscat2, Iodine, and Ozyman in our experiments to generate malicious network traffic. These tools generate data exfiltration and command and control (C&C) traffic. These tools operate in different ways; for example, they use different encoding, client–server architecture, etc. This helps create a variety of malicious network traffic in our datasets. The benign network packets were generated by capturing network traffic on a corporate network at geographically different offices in Canada and the US. The packets were captured over a period of 5 weeks for a total of 1.5 GB of data. We use a portion of this data using random sampling to create our benign dataset.

Feature selection

Most of the time, the raw network packets that are captured for anomaly detection have data that are not *informative*. For instance, date, time, and even IP address are not providing any information about the nature of the traffic, as they can freely change over time.

One of the most important steps before training a machine learning model is *Feature Engineering*. This step involves selecting the features that have *Predictive Power*. In our scenario, these are the features that have correlation to our target classes; malicious or benign.

Table 1 shows the original features that we extracted from the captured network packets. One initial processing that often needs to be performed on datasets is data type conversion. Many machine learning algorithms work only with numerical features, so other data types must be converted to have a numerical representation. This can be done in different ways, for example a naive way to do so would be $[c1, c2, c3, \ldots] \; => \; [n1, n2, n3, \ldots]$ where the first vector represents the categorical values, and the second one is corresponding numerical values. The issue with this type of conversion is that each categorical value is being assigned to a numerical value with a different size, and this could cause bias in some learning algorithms.

Data transformation

We use one-hot encoding [24] on categorical features. In this method of encoding, a categorical feature will be converted into an array of features each taking only 0 or 1 as values. For example, if we have a categorical feature F with three different values v_1, v_2, and v_3, then we will replace feature F with three new features F_1, F_2, and F_3. If an example originally had the value v_1, it will now have the values 1, 0, 0 for F_1, F_2, and F_3, respectively. This ensures that all the categorical values are treated the same, and there is no bias introduced because of this conversion.

Other transformations that are usually used on the dataset are normalization and standardization scaling. The two techniques are similar in the way that they both limit the range of values for each feature in the dataset. However, normalization usually converts the values to be in the range [0, 1], whereas standardization or z-score normalization converts the feature values to have a *standard normal distribution* with $\mu = 0$ and $\sigma = 1$. Formulas 1 and 2 show how these transformations are done on a feature called X

$$X_{\text{standard}} = \frac{X - \mu}{\sigma} \tag{1}$$

$$X_{\text{normal}} = \frac{X - X_{\min}}{X_{\max} - X_{\min}}. \tag{2}$$

Most classification algorithms benefit from feature scaling, but some such as k-NN, SVM, and MLP that are distance-based benefit more. However, even tree-based classifiers such as decision trees and Random Forest benefit from scaling, as well.

Figure 3 shows the original feature set's correlation heat map. A darker red means more positive correlation to the *label*, whereas a lighter mean more negative correlation. We are looking for the features that are very positively or very negatively correlated to the target. A positive correlation between two variables means that they move in the same direction (if one increases the other increases or decreases as the other decreases). On the other hand, a negative correlation means that the two variables move in different directions; as once increases, the other decreases and vice versa. We select a feature x, such that $|Corr_x| \geq 0.4$, where $Corr_x$ denotes the correlation of feature x to the label.

Table 2 shows the selected features. We use Pearson Correlation Coefficient (PCC) [25] values to choose our feature set. PCC evaluates the value of an attribute by measuring its correlation to the class. The higher the correlation, the better that feature represents the corresponding class.

Experimental results

We use scikit-learn [7] to implement our models in Python. This popular and widely used machine learning library provides APIs to implement the most common algorithms in an

Table 1 All captured network packet fields

Feature	Type	Description
Question type	Categorical	DNS query question type such as A, CNAME, MX, etc.
Is Standard query	Boolean	Whether the query is a standard DNS query
Is Authoritative	Boolean	Whether the name server is authoritative for the query
Is Reply success	Boolean	Whether there is an error in the query response
Is Reverse lookup	Boolean	Whether this is a reverse DNS lookup
Round trip time	Numeric	Round trip time (RTT) for the packet
Request answer count	Numeric	Number of resource records in the answer section for request
Request authority count	Numeric	Number of name server resource records in the authority records section for request
Request Additional count	Numeric	Number of resource records in the additional records section for request
Question length	Numeric	Length of the DNS query question in bytes
Question info bits	Numeric	Synthetic feature estimating number of bits to encode a DNS question
Question entropy	Numeric	Synthetic feature measuring the entropy of the DNS question
Response answer length	Numeric	Length of the DNS response answer in bytes
Response answer info bits	Numeric	Synthetic feature estimating number of bits to encode DNS response answer
Response answer count	Numeric	Number of resource records in the answer section for response
Response additional count	Numeric	Number of resource records in the additional records section for response
Response authority count	Numeric	Number of name server resource records in the authority records section for response
Response question count	Numeric	Number of questions in the response
Response answer TTL	Numeric	Time to live for response answer section
Response answer length	Numeric	Length of answer section in response
Response authority length	Numeric	Length of authority section in response
Response authority info bits	Numeric	Synthetic feature estimating number of bits to encode response authority
Response additional length	Numeric	Length of additional records section in response
Response additional info bits	Numeric	Synthetic feature estimating number of bits to encode additional records section in response
Response info bits	Numeric	Synthetic feature estimating number of bits to encode a DNS response
Response entropy	Numeric	Synthetic feature measuring entropy of the DNS query response

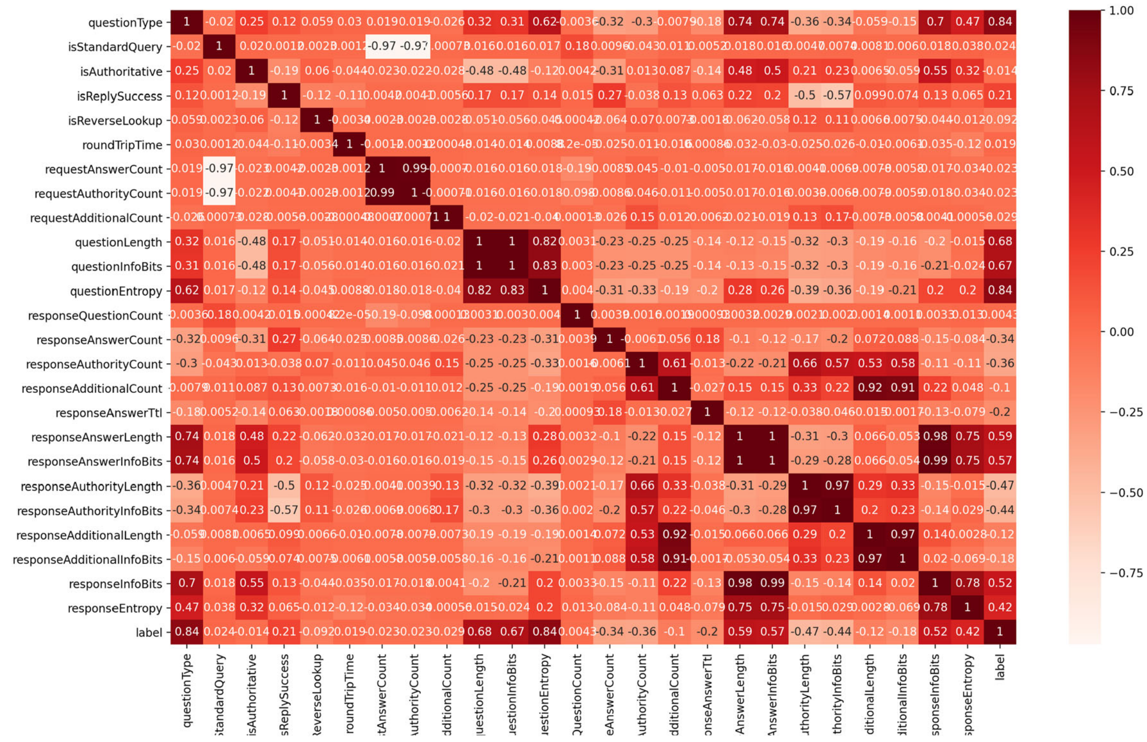

Fig. 3 Features heat map. The darker means more correlation to the label (class). Darker red means more positive correlation, and lighter red means more negative correlation

Table 2 Selected features with a correlation (≥ 0.4)

Feature	Correlation
Question Type	0.843047
Question Length	0.675164
Question Info Bits	0.668894
Question Entropy	0.839315
Response Answer Length	0.590283
Response Answer Info Bits	0.574044
Response Authority Length	0.466657
Response Authority Info Bits	0.439765
Response Info Bits	0.521971
Response Entropy	0.422856

efficient way. We run our experiments on a MacBook Pro with a 2.6 GHz 6-Core Intel Core i7 processor, and 6 GB 2400 MHz DDR4 memory on macOS Catalina. We discuss how bagging, boosting, and stacking ensemble learners compare against each other and other classifiers in detecting low footprint network intrusions.

We do not use cross validation or hold-out sets to measure the performance of our models. Training and testing on different portions of the same dataset are not reliable in network intrusion detection [22]. These validation techniques could cause biased results; as such, we validate our model on a dataset that has different characteristics. This means that the test dataset has examples that have been generated using different tools, or captured on geographically different network locations. This ensures that not only the test data have different examples, but they also have different characteristics.

Hyperparameter tuning

We use *Grid Search* for hyperparameter tuning to find the set that achieves the best F1 score. This metric performs a better evaluation of a model when false-positive and false-negative rates are more important. On the other hand, if true-positive and true-negative rates are crucial, hyperparameters can be tuned to achieve the highest accuracy instead. In case of a tie for F1 scores between two or more hyperparameters, we choose the one that has the lowest false-positive rate. This would help choose the model that is more useful in practice. Avoiding false positives or false alarms is critical in a real-world intrusion detection system.

Performance metrics

Formula 3 shows the metrics used to measure the performance of each model in classifying the low footprint network intrusions. The two mostly used metrics in general are *Accuracy* and *F1 score*. Accuracy is used when true positives and

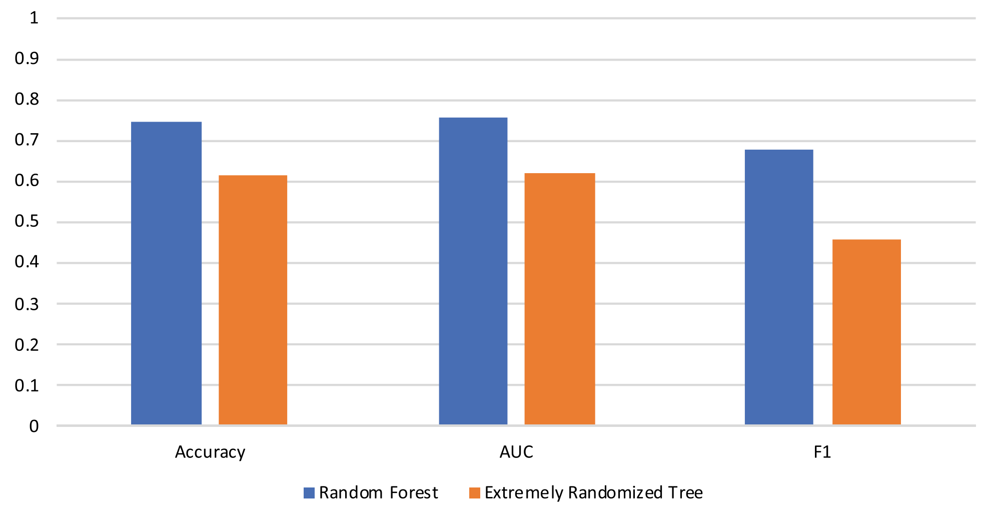

Fig. 4 Bagging performance for two different techniques: Random Forest vs. Extremely Randomized Trees. Random Forest outperforms in all three performance metrics

true negatives are more important, whereas F1 score is used when false positives and false negatives matter the most. In intrusion detection, a false alarm is the type of error that the system strives to avoid; thus, F1 score is a more important metric here. We do not report Precision or Recall, as F1 score is the harmonized mean of the two

$$\begin{aligned}
\text{Precision} &= TP \times (TP + FP)^{-1} \\
\text{Recall} &= TP \times (TP + FN)^{-1} \\
\text{Accuracy} &= (TP + TN) \times (TP + TN + FP + FN)^{-1} \\
F1 &= 2 \times (\text{Precision} \times \text{Recall}) \\
&\quad \times (\text{Precision} + \text{Recall})^{-1} \\
AUC &= \int_{x=0}^{1} TPR(FPR^{-1}(x))\mathrm{d}x. \qquad (3)
\end{aligned}$$

Bagging performance

We use Random Forest and Extremely Randomized Trees [26] as our bagging ensemble classifiers. An Extremely Randomized Tree is very similar to a Random Forest in terms of building multiple trees for the model. However, extremely randomized trees do not use bootstrapping by default, which means that they sample the dataset without replacement. The other difference between the two is that splitting a parent node into child nodes is based on best split in Random Forest, but is random in extremely randomized trees.

Figure 4 shows the classification performance of the two bagging algorithms. These are the best performances achieved by each classifier based on a Grid Search. As the figure shows, Random Forest outperforms Extremely Randomized Tree in every metric. However, neither of the bagging ensemble classifiers has a high F1 score or accuracy. More specifically, Extremely Randomized Tree's very low F1 score is due to its very high false-negative rate.

Boosting performance

We use three boosting classifiers: Gradient Boost, Ada Boost, and XGBoost. Figure 5 shows how the three different algorithms perform on the low footprint intrusion dataset. As the figure shows, they all have very similar performance with no

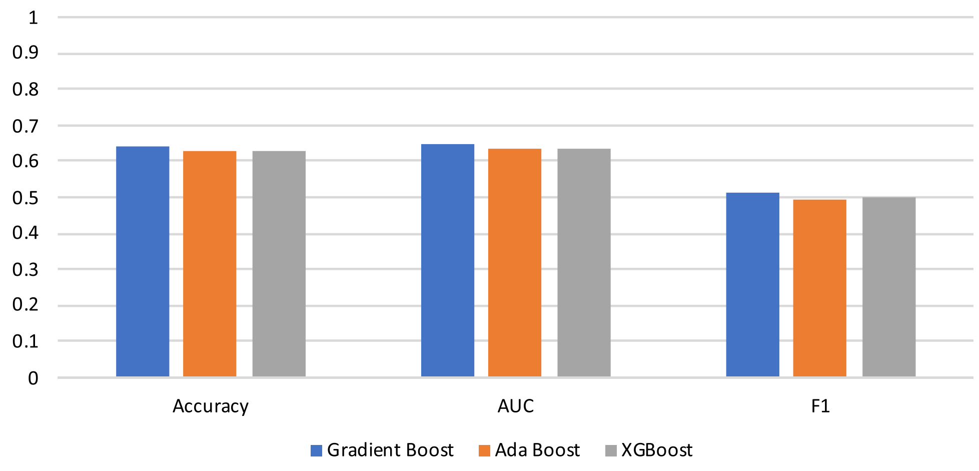

Fig. 5 Boosting performance for three techniques: Gradient boost, Ada boost, and XGBoost

Table 3 List of all the classifiers used in building stacking ensemble learners along with their hyperparameters

Classifier	Hyperparameters tuned
k-nearest neighbors	n_neighbors, metric, weights
Support vector machine	Kernels, C, penalty
Decision tree	max_depth
Stochastic gradient Descent	Alpha
Naive bayes	Distribution
Logistic regression	Solver, penalty, C
Multi-layer perceptron	Activation, hidden_layer_sizes, solver, alpha, learning_rate

one outperforming the others. F1 scores are approximately 0.5, and the Accuracy and AUC hover at 0.64. Similarly to the bagging classifiers, the boosting ensemble classifiers perform poorly in classifying the low footprint intrusions.

Stacking performance

We evaluate two different stacking techniques; simple and multi-layer. The former consists of only one meta classifier, whereas the latter uses multiple meta classifiers on different layers. For a simple stacking learner, the architecture is limited to the choice of classifiers on the first layer, and the meta classifier on the second (last) layer. However, there are many different architectures to choose from in a multi-layer stacking model.

Table 3 shows all the classifiers used to build the stacking ensemble learners and their hyperparameters. These hyperparameters are tuned via Grid Search to find the best-performing model for each classifier.

Figure 6 shows the performance of individual classifiers on the low footprint network intrusions dataset. Except decision tree and k-NN, the other classifiers have high accuracy, AUC, and F1 scores. These are the best results for each classifier based on Grid Search.

Table 4 shows the performance of single-layer stacking models for the best two-, three-, and four-learner models. All the models have very high accuracy, AUC, and F1 scores. The false-positive rates are very low too. In general, the three-learner had slightly better performance than the two- or four-learner models, especially in terms of false-positive rate, which is crucial in intrusion detection. This experiment also shows that having more base learners does not necessarily improve the performance, as the best four-learner model has a lower performance compared to the best three-learner model. However, the latter performs better than a two-learner model.

Performance acceptance criteria

Our acceptance criteria for an ensemble classifier that can effectively detect low footprint network intrusions are as follows:

$$\begin{aligned} &F1\_\text{score} \geq 0.99 \\ &\text{Accuracy} \geq 0.99 \\ &AUC \geq 0.99 \\ &FPR \leq 0.001. \end{aligned} \tag{4}$$

Our goal is to design a stacking model (single or multi-layer) that satisfies the criteria in Eq. 4. From all the different stacking models that we created, the ensemble models that combined Stochastic Gradient Descent (SGD), Decision Tree (Tree), Naive Bayes (NB), and Logistic Regression (LR)

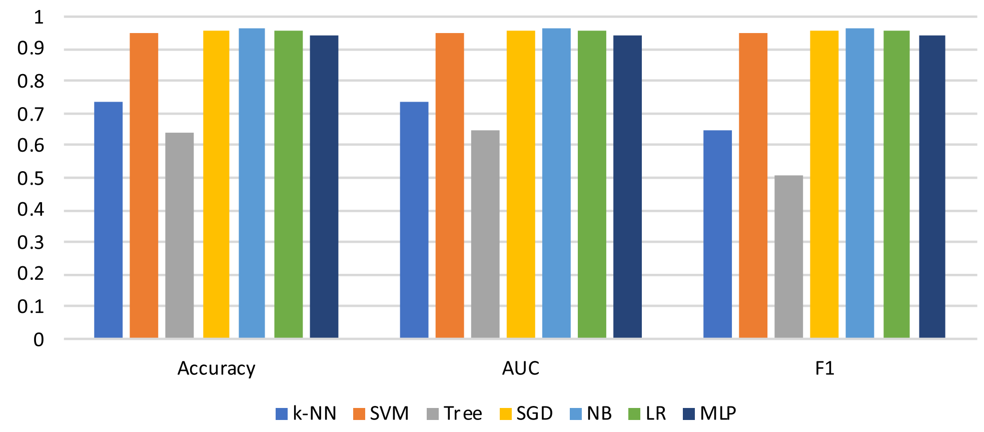

Fig. 6 Individual classifier performances on the dataset. Most single classifiers have very high accuracy, AUC, and F1 scores. k-NN and decision tree are lagging behind others in terms of performance. They are too simple to be able to create robust models in this scenario

Table 4 Top-performing single-layer ensemble models

Meta classifier	Learners	Accuracy	AUC	F1	FPR
MLP	NB + SGD	0.978	0.978	0.978	0.005
MLP	SVM + NB + SGD	0.994	0.994	0.994	0.004
MLP	k-NN + SVM + NB + SGD	0.987	0.987	0.987	0.006

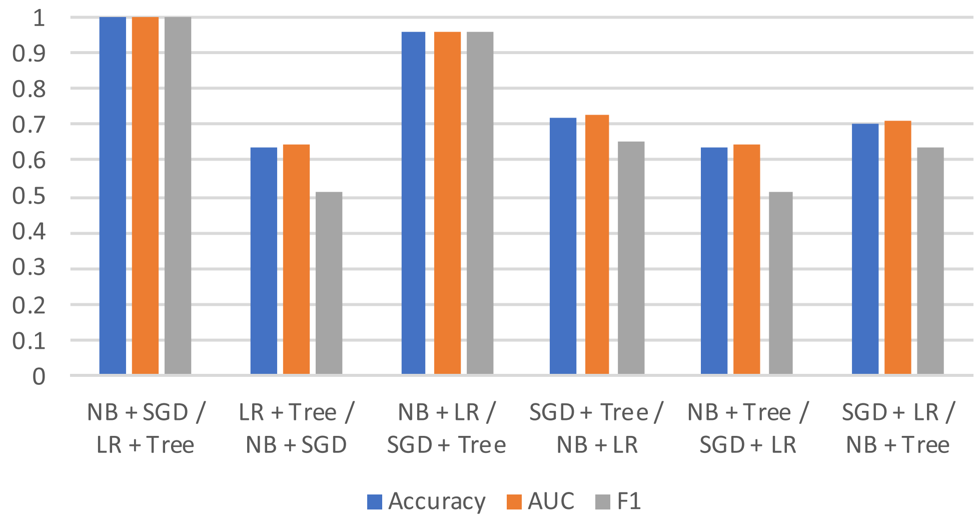

Fig. 7 Different multi-layer stacking models based on different placement of the classifiers. The best stacking ensemble models were created by placing NB and SGD on the first layer and LR and Tree on the second layer with NLP as the meta classifier

achieved the highest performance in terms of accuracy, AUC and F1 scores. Nevertheless, the layer where every single classifier is placed plays a big role in the performance of the final stacking ensemble model. Figure 7 shows all the six possible stacking ensemble models created from the four classifiers that built the best model. The best overall stacking ensemble was created by placing NB and SGD on the first, and LR and Decision Tree on the second layer. This combination resulted in an almost perfect model with F1 score, accuracy and AUC of 0.99, and FPR of 0.001. On the other hand, placing LR and Decision Tree on the first layer, and NB and SGD on the second layer resulted in a model with poor performance

Table 5 shows all the stacking ensemble models that achieved the performance criteria (Eq. 4). Only multi-layer stacking ensemble models were able to produce such high performance models. Neither single-layer stacking models, nor bagging or boosting models satisfied the performance criteria. Figure 8 shows the variations in stacking ensemble models F1 scores when the same base learners are placed on different layers. As the figure shows, the F1 score varies

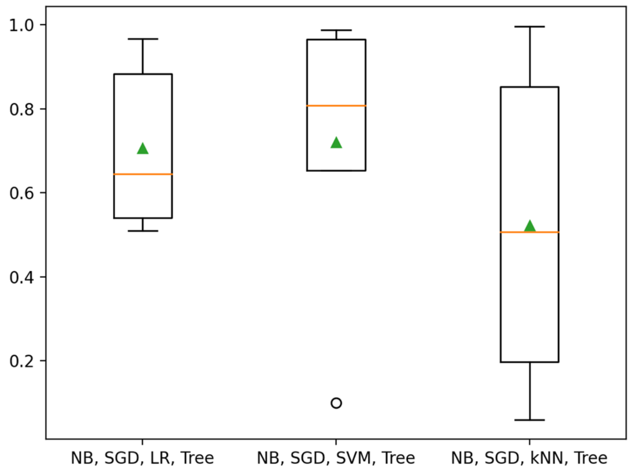

Fig. 8 Variations in stacking ensemble F1 score when the same base learners are placed on different layers

from less than 0.1 for some outlier models to 0.99 for the top-performing models.

Figure 9 shows the impact of using different meta classifiers when the learners on each layer are unchanged. The

Table 5 The only ensemble models satisfying the performance criteria

Meta classifier	Layer1 learners	Layer2 learners	Accuracy	AUC	F1	FPR
MLP	NB + SGD	LR + Tree	0.999	0.999	0.999	0.001
MLP	NB + SGD	SVM + Tree	0.998	0.998	0.997	0.001
MLP	NB + SGD	k-NN + Tree	0.998	0.998	0.998	0.001

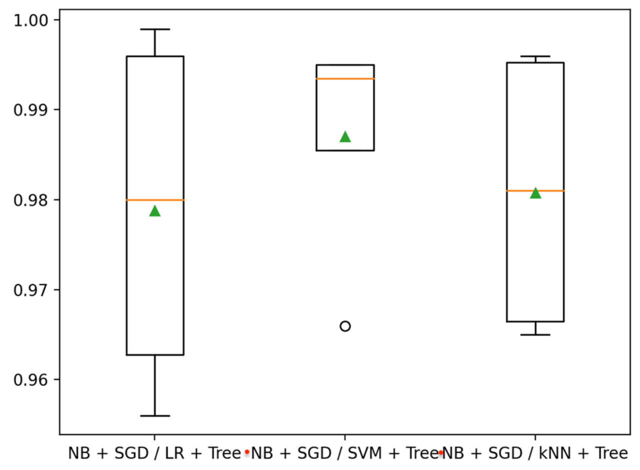

Fig. 9 Variations in stacking ensemble F1 score when different meta classifier used, while base learners kept unchanged

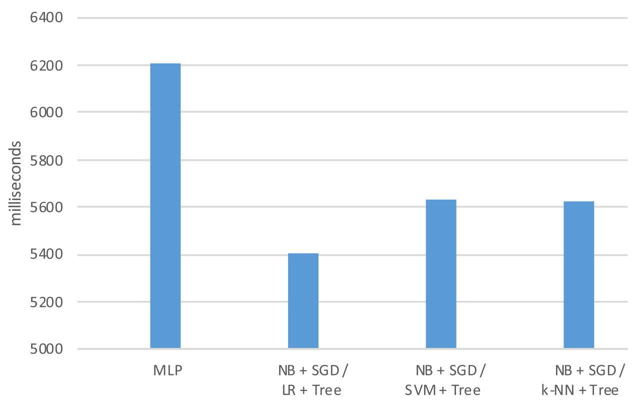

Fig. 10 Training times between MLP and the top-performing multi-layer ensemble models. The MLP has one hidden layer of size 100. It is faster to train the ensemble model than a simple MLP

learners are those that had the best performance (Table 5). As the box plot shows, there is a noticeable difference in F1 score between the best meta classifier (MLP) and others such as logistic regression. Moreover, the very low false-positive rate of 0.001 was only achievable when MLP was used as the meta classifier.

Another observation from creating a stacking ensemble model is that a classifier that performs poorly as an individual learner may do well in an ensemble model. In our experiments, Decision Tree did not perform well (Fig. 6) individually. However, this classifier performed better when used as a base learner in combination with other learners. In fact, it was a part of all the three ensemble models that satisfied the performance criteria (Table 5). The role that the simple Decision Tree plays in the stacking ensemble is to help reduce the false-positive rate. This then leads to increasing F1, accuracy, and AUC.

Figure 10 shows the time taken to train an MLP model versus each of the top-performing multi-layer ensemble models. The MLP model has only one hidden layer of size 100. As the figure shows, it is quite faster to train the multi-layer ensemble models than training a basic MLP. The difference in training speed is mostly due to the higher complexity of the MLP compared to the other models. More specifically, the relative slowness is caused by the number of multiplications required to compute the activation of all the neurons in the neural network.

Deep neural networks

To compare the classification performance of stacking ensemble models with deep neural network models, we implemented a Deep Neural Network (DNN) and a Convolutional Neural Network (CNN) model. We used PyTorch [27] to implement the deep learning models. We used four linear layers on the DNN and four convolution layers on the CNN model. We train both models using batching and apply Batch Normalization. We used ReLU activation function and did not use Dropout during training of final models as it reduced the classification performance. Both models use backpropagation and employ the BCEWithLogitsLoss loss function.

We tuned both models by evaluating different network topologies, number of epochs, batch sizes, and optimizers. Table 6 shows the classification performance of the best DNN and CNN models along with their parameters. As the table shows, both DNN and CNN models had a very similar and rather high classification performance. Nonetheless, these deep learning models were outperformed by out stacking ensemble models that had accuracy and F1 scores of higher than 0.99.

Conclusion

In this paper, we show that stacking ensemble models that combine simple base learners can outperform many other

Table 6 Deep learning models' parameters and classification performance

Model	# Layers	Batch size	# Epochs	Optimizer	Accuracy	F1
DNN	4 (Linear)	64	50	SGD	0.9634	0.9633
CNN	4 (Conv1d)	128	15	Adam	0.9651	0.9650

models in detecting low footprint network intrusions. We show that stacking techniques are superior to bagging and boosting methods. We further show that multi-layer stacking performs better than single-layer stacking.

We introduce performance acceptance criteria that require very high accuracy, AUC, and F1 score and a very low false-positive rate. We show that among hundreds of potential ensemble models that can be created using our seven base learners, only three multi-layer models meet our performance criteria. These models have naive Bayes and stochastic gradient descent on the first, and two of logistic regression, support vector machine, k-NN, and decision tree on their second layers. All these top models use multi-layer perceptron as the final meta learner. These models achieved accuracy, AUC, and F1 score of higher than 0.99 and false-positive rates of as low as 0.001.

Another interesting result from our experiments is that there is a *sweet spot* for a single-layer stacking in terms of the number of learners. Based on our experiments, combining three learners leads to the best results, and adding or removing learners will reduce the performance. If we remove learners, underfitting will happen. On the other hand, if we add learners, overfitting will occur. Since the test dataset has different types of network intrusion samples, a model overfitted (or underfitted) on the training set would not perform as well as a "balanced" mode. The other key finding is that placing the learners on the right layer plays an important role in the classification performance. We show that the F1 score can vary more than 80% between two multi-layer stacking models when the same base learners are placed on different layers. Finally, we show that our stacking ensemble models outperform deep learning models in classifying low footprint intrusions.

Nevertheless, there are limitations in creating stacking ensemble models that achieve very high classification performance results. One drawback of these ensemble models is that they have a bigger memory and processor footprint compared to some non-ensemble models. This could be important where real-time traffic is being classified, and thus, the model is required to run on a more powerful machine. However, based on our experiments, a multi-layer stacking ensemble is still faster to train and test examples than a feedforward neural network such as MLP with only one hidden layer. In industrial intrusion detection systems, there is usually little tolerance for false positives (false alarms). As a result, if achieving a very high F1 score and near-zero false-positive rates is desired, then this drawback could be traded off with the high performance.

Funding No funding was received to assist with the preparation of this manuscript.

Declarations

Conflicts of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Sommer R, Paxson V (2010) Outside the closed world: On using machine learning for network intrusion detection. In: IEEE symposium on security and privacy. IEEE 2010, p. 305–316
2. Breiman L (2001) Random forests. Mach Learn 45(1):5–32
3. Chen T, Guestrin C (2016) Xgboost: A scalable tree boosting system. In: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, p. 785–794
4. Archive UK (1999) Kdd cup 1999 dataset. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. Accessed 19 February 2022
5. Tavallaee WLM, Bagheri E, Ghorbani A (2009) Nsl-kdd dataset. https://www.unb.ca/cic/datasets/nsl.html. Accessed 30 March 2022
6. Young S, Abdou T, Bener A (2018) Deep super learner: A deep ensemble for classification problems. In: Canadian Conference on Artificial Intelligence. Springer, p. 84–95
7. 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 (2011) Scikit-learn: machine learning in Python. J Mach Learn Res 12:2825–2830
8. Aburomman AA, Reaz MBI (2017) A survey of intrusion detection systems based on ensemble and hybrid classifiers. Comput Secur 65:135–152
9. Vanerio J, Casas P (2017) Ensemble-learning approaches for network security and anomaly detection. In: Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks, p. 1–6
10. Folino G, Sabatino P (2016) Ensemble based collaborative and distributed intrusion detection systems: a survey. J Netw Comput Appl 66:1–16
11. Syarif I, Zaluska E, Prugel-Bennett A, Wills G (2012) Application of bagging, boosting and stacking to intrusion detection. In: International Workshop on Machine Learning and Data Mining in Pattern Recognition. Springer, p. 593–602
12. Gu J, Wang L, Wang H, Wang S (2019) A novel approach to intrusion detection using svm ensemble with feature augmentation. Comput Secur 86:53–62

13. Shafieian S, Zulkernine M, Haque A (2015) Cloudzombie: Launching and detecting slow-read distributed denial of service attacks from the cloud. In: 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing. IEEE, p. 1733–1740
14. Shafieian S, Smith D, Zulkernine M (2017) Detecting dns tunneling using ensemble learning. In: International Conference on Network and System Security. Springer, p. 112–127
15. Gao X, Shan C, Hu C, Niu Z, Liu Z (2019) An adaptive ensemble machine learning model for intrusion detection. IEEE Access 7:82 512-82 521
16. Hsu Y-F, He Z, Tarutani Y, Matsuoka M (2019) Toward an online network intrusion detection system based on ensemble learning. In: 2019 IEEE 12th International Conference on Cloud Computing (CLOUD). IEEE, p. 174–178
17. Moustafa N, Slay J (2015) Unsw-nb15: a comprehensive data set for network intrusion detection systems (unsw-nb15 network data set). In Military Communications and Information Systems Conference (MilCIS) 2015, p. 1–6
18. Zhong Y, Chen W, Wang Z, Chen Y, Wang K, Li Y, Yin X, Shi X, Yang J, Li K (2020) Helad: a novel network anomaly detection model based on heterogeneous ensemble learning. Comput Netw 169:107049
19. Mawilab dataset. http://www.fukuda-lab.org/mawilab/index.html. Accessed 6 April 2022
20. Cic-ids2017 dataset. https://www.unb.ca/cic/datasets/ids-2017.html. Accessed 6 April 2022
21. Mirsky Y, Doitshman T, Elovici Y, Shabtai A (2018) Kitsune: An ensemble of autoencoders for online network intrusion detection. In: 25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18–21, 2018
22. Tama BA, Comuzzi M, Rhee K-H (2019) Tse-ids: A two-stage classifier ensemble for intelligent anomaly-based intrusion detection system. IEEE Access 7:94 497–94 507
23. Mirza AH (2018) Computer network intrusion detection using various classifiers and ensemble learning. In: 26th Signal Processing and Communications Applications Conference (SIU). IEEE 2018:1–4
24. One-hot encoding. https://en.wikipedia.org/wiki/One-hot. Accessed 30 April 2022
25. Pearson correlation coefficient. https://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient. Accessed 4 February 2022
26. Geurts P, Ernst D, Wehenkel L (2006) Extremely randomized trees. Mach Learn 63(1):3–42
27. Pytorch machine learning framework. https://pytorch.org. Accessed 7 May 2022