*Journal of AppliedMath 2023; 1(4): 271.*
Original Research Article

# Development of a stacked hybrid Decision Tree model leveraging the NSL-KDD dataset

**Edosa Osa[1,*], Patience Orukpe[1], Iruansi Usiholo[2]**

[1] *Department of Electrical and Electronic Engineering, Faculty of Engineering, University of Benin, P.M.B. 1154, Benin City, Nigeria*

[2] *Department of Computer Engineering, Faculty of Engineering, University of Benin, P.M.B. 1154, Benin City, Nigeria*

*** Corresponding author:** Edosa Osa, edosa.osa@uniben.edu

**ARTICLE INFO**

Received: 19 October 2023
Accepted: 7 December 2023
Available online: 21 December 2023

doi: 10.59400/jam.v1i4.271



**ABSTRACT:** Intrusion detection in information technology as well as operational technology networks is highly required in modern day systems due to the increased spate of cyber-attacks in both number and complexity. Anomaly-based intrusion detection systems which have the capacity to detect novel or zero-day attacks are highly employed in this regard. One important component of anomaly-based intrusion detection systems which ensures their behaviour is artificial intelligence in general and machine learning in particular. The burden in modern day cybersecurity research is to investigate and develop models that can outperform existing ones. This paper is aimed at developing a hybrid decision tree model using the stacking ensemble approach. Performances were measured on the basis of recall, precision, accuracy, F1-score, receiver operating characteristics and confusion matrices. The hybrid model presented a precision of 97%, accuracy of 81%, F1-score of 80% and AUC score of 0.96, respectively.

***KEYWORDS:*** machine learning; decision tree; hybrid; intrusion; detection

## 1. Introduction

The advent of the third industrial revolution saw the rise of computers as well as internet technology. This development came with many potential enhancements for existing human engagements[1]. Thus, many processes that used to involve human contact now are based on both human and electronic interaction or are entirely digital. Governance for instance can now be electronic hence the term e-governance, and banking does not always have to take place within the premises of a banking hall since financial transactions could be carried out by simply pressing buttons or manipulating interfaces of digital devices that are connected to relevant networks, among other benefits. These developments are however shadowed by the prevalence of cybercrime in computer networks and the Internet. Malicious actors probe such networks on a constant basis seeking for easy targets so as to deploy various threats and attacks in compromising the safe and efficient functioning of computer networks[2]. Such threats and attacks include worms, viruses, denial of service (DoS) attacks, distributed denial of service (DDoS) attacks, trojan, injection attacks, and the like[3,4] which tend to compromise the confidentiality, integrity and availability of networks[5–7]. One peculiar intervention being employed to defend networks is the intrusion detection system (IDS). An intrusion detection system is a software and hardware system combination that is implemented to detect intrusions or unauthorized access into computer networks[8,9]. Such a system operates by scanning traffic in order to detect unauthorized access and then report such, based on

preconfigured detection parameters[10]. IDS classifications on basis of habitation scope are Host-based Intrusion Detection Systems (HIDS) and Network Intrusion Detection Systems (NIDS). A host-based system could be a single workstation which monitors very crucial operating system files and protects such files from attack. A NIDS operates by analyzing designated network traffic so as to detect malicious signatures in the network[11]. IDSs could also be classified according to their detection approach, into Signature-based IDS (SIDS) and Anomaly-based IDS (AIDS). A Signature-based IDS carries out detection by searching for patterns in the traffic such as byte sequences or instruction sequences and matches the discovered patterns against a pre-existing signature database[12]. SIDs however are limited in detecting novel or zero-day attacks[12]. Anomaly-based intrusion detection systems (AIDSs) on the other hand are capable of monitoring network traffic and making continuous comparison with an established normal behaviour baseline. Such systems detect anomalous traffic across the network plane, in devices, ports, protocols, bandwidth, etc.[13]. Highly efficient anomaly-based intrusion detection systems (AIDSs) are developed by implementing machine learning which is the process of programming or instructing computing devices so that they can learn from data[14]. This approach to intrusion detection involves training relevant machine learning algorithms to develop models of trustworthy activity which could be deployed for intrusion detection. Such AIDSs can be used to expose evolving threats and zero-day attacks thereby overcoming a major limitation of the signature-based system[13]. The NSL-KDD dataset is a popular benchmark in the machine learning intrusion detection ecosystem and several authors have employed this dataset for binary classification problems (i.e., normal or attack traffic). In the work by Alladi[15], a hybrid classifier model for intrusion detection which was composed of the K-means clustering algorithm and the neural network Multilayer Perceptron algorithm was investigated and an accuracy of 73.09% was obtained. In the work by Sapre et al.[16], the authors compared the performances of KDD'99 and NSL-KDD datasets with four classifiers, namely Support Vector Machine, Artificial Neural Network, Random Forest and Naïve Bayes classifier. For NSL-KDD, their approach resulted in accuracy values of 71.32%, 78.51%, 74.41% and 61.08% for Support Vector Machine, Artificial Neural Network, Random Forest and Naïve Bayes, respectively. By leveraging on Apache Spark, Vinayakumar et al.[17] proposed Deep Neural Network (DNN) for intrusion detection systems. Their model achieved an accuracy score of 79.42% when tested with the NSL-KDD test dataset. However, more efficient AIDs are required in order to improve performance and reduce the number of false positives during prediction by the machine learning models. Ensemble techniques such as stacking could be implemented to develop a hybrid approach to intrusion detection where disparate machine learning models are stacked together so as to underline new and improved anomaly-based intrusion detection systems for better performance. Hence, this work employs a stacking approach to improve on the prediction level of the decision tree model when used to fit the NSL-KDD Dataset. The aim of this work was achieved, since a hybrid classifier model capable of improved prediction and lower number of false positives was developed.

## 2. Materials and methods

### 2.1. Experimental environment

The experimental environment for this research was the Google Colaboratory environment in which a single Jupyter notebook was populated with the necessary python codes. The operating system provisioned was Linux-5.15.107+-x86_64-with-glibc2.31. Libraries that were provisioned include Python 3.10.12, Pandas 1.5.3, NumPy 1.22.4, Seaborn 0.12.2, SciPy 1.10.1 and Scikit-Learn 3.7.1. The flow diagram for the hybrid model design is presented in **Figure 1**.

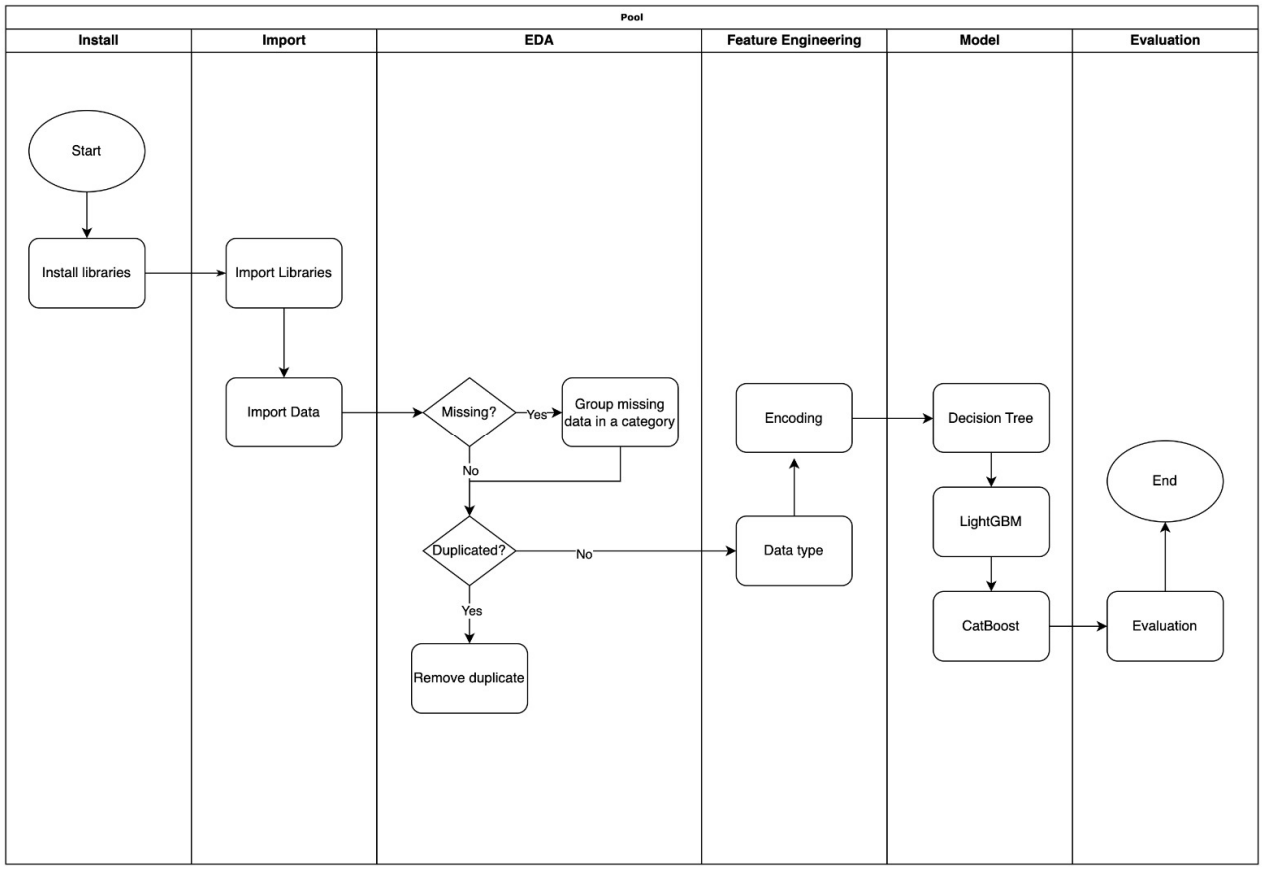


**Figure 1.** Flow diagram of hybrid model design.

## 2.2. Install and import

The necessary python packages for the Catboost and LightGBM classifiers were installed onto the Jupyter notebook environment. Also, the necessary software libraries for developing the machine learning models such as pandas for data manipulation, seaborn with matplotlib for visualization as well as numpy for numerical processes were imported at this point in the experiment. Libraries for data preprocessing such as StandardScaler, MinMaxScaler and OneHotEncoder were also imported. The machine learning algorithms employed in this work were imported from scikit-learn collection, namely DecisionTreeClassifier, LGBMClassifier and CatboostClassifier. Evaluation metrics for the developed models such as accuracy_score, confusion_matrix, recall_score, precision_recall_curve, auc_roc_curve, roc_auc_score, classification_report, and RocCurveDisplay were also imported. The training data, i.e., NSL-KDD Train+ which consists of 125,973 samples and the NSL-KDD test dataset which consists of 22544 samples respectively were imported as well. These datasets were adopted from the official website of the University of New Brunswick, Canada.

## 2.3. Exploratory data analysis (EDA)

Exploratory data analysis was carried out to provide insight into the provided dataset so as to ascertain readiness for implementation in building the classifier models. The data was first described via the df train.info() function and thereafter the datatypes pertaining to the various feature columns in the NSL-KDD trainset+ were investigated. The datatype (with number of associated features) distribution contained in the dataset included: float64 (15), int64 (24), object (4). The presence of missing values in the training dataset was investigated next via the df_train.isna().sum() function. The presence of missing

values in the dataset can affect the viability of the dataset when fitting is done by machine learning algorithms. Upon investigation, both training and test datasets contained no missing values. The next step was investigation for the presence of duplicate values in the training dataset that could also hinder the efficiency of the developed models, no duplicate values were found. This was carried out via the df_train.duplicated().sum() function. During the exploration of the dataset, outliers were discovered in the src_bytes and destination_bytes columns as shown in **Figure 2**.

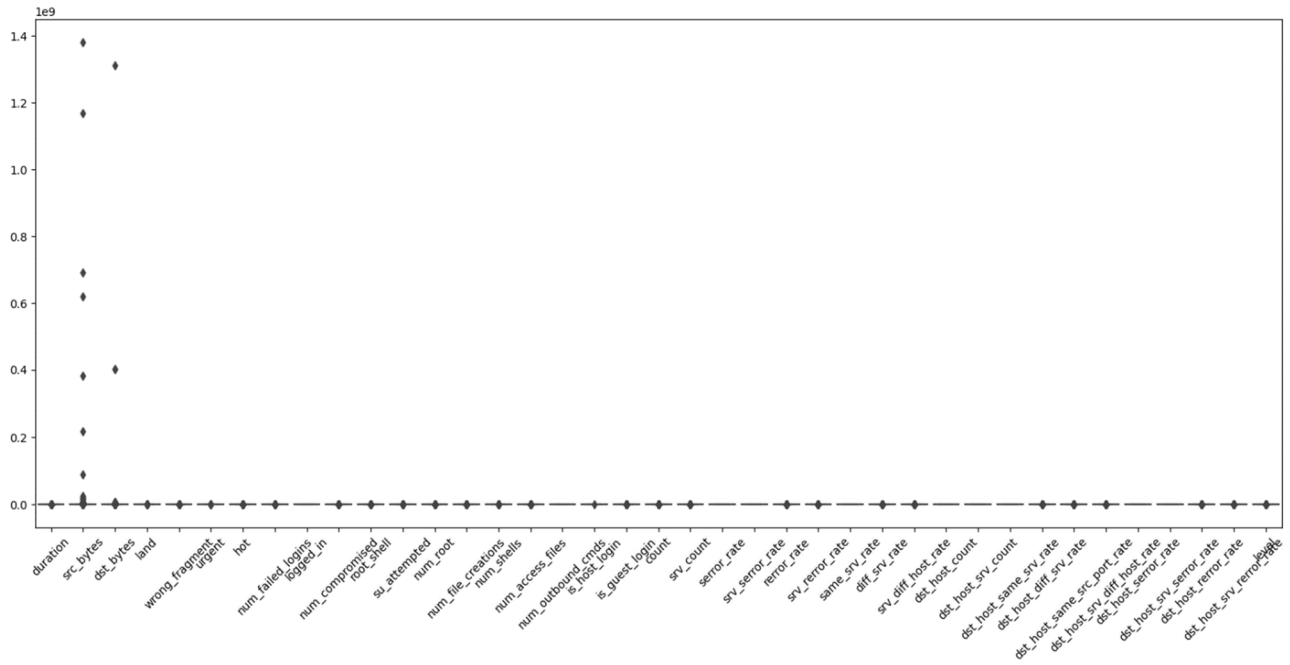


**Figure 2.** Distribution of outliers in training data.

## 2.4. Feature engineering

The prime deed at this stage was conversion of categorical variables represented with other datatypes to the 'category' datatype by employing the relevant function as shown in **Figure 3**.

```
cat_cols = ['protocol_type','service','flag','land','logged_in','root_shell','su_attempted','is_host_login','is_guest_login','attack']
```

```
for feature in cat_cols:
    df_train[feature] = pd.Series(df_train[feature], dtype="category")
    df_test[feature] = pd.Series(df_test[feature], dtype="category")
```

**Figure 3.** Conversion of categorical variables to 'category'.

The following columns affected by the conversion include 'protocol_type', 'service', 'flag', 'land', 'logged_in', 'root_shell', 'su_attempted', 'is_host_login', 'is_guest_login' and 'attack'. This engineering process resulted in a new distribution for the training dataset.

The target label (i.e., attack) was also changed to binary form in order to realize the binary classification task. The four attack types present in the dataset (i.e., DOS, U2L, R2L and Probe) were grouped under one label (i.e., attack) against the normal label. The normal label was encoded with the binary value of "0" while attack label was encoded with the binary value of "1". Further encoding during feature engineering addressed the skew present in the raw dataset (shown in **Figure 4**). A sufficient level of balance was obtained with 58,630 features for attack traffic and 67,343 features for normal traffic respectively.

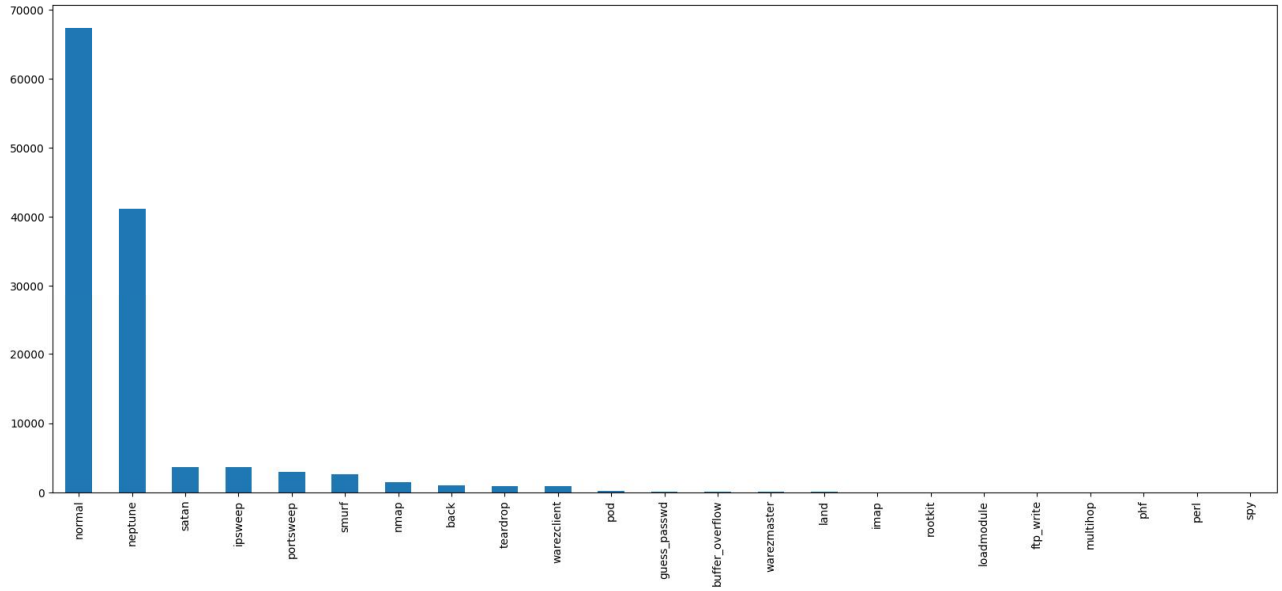


**Figure 4.** High skew of attack attribute in raw dataset.

Feature engineering resulted in a new target label dubbed 'attack_binary'. The new data distribution after encoding is described by the pie chart in **Figure 5,** where 47% = 58,630 and 53% = 67,343, respectively.

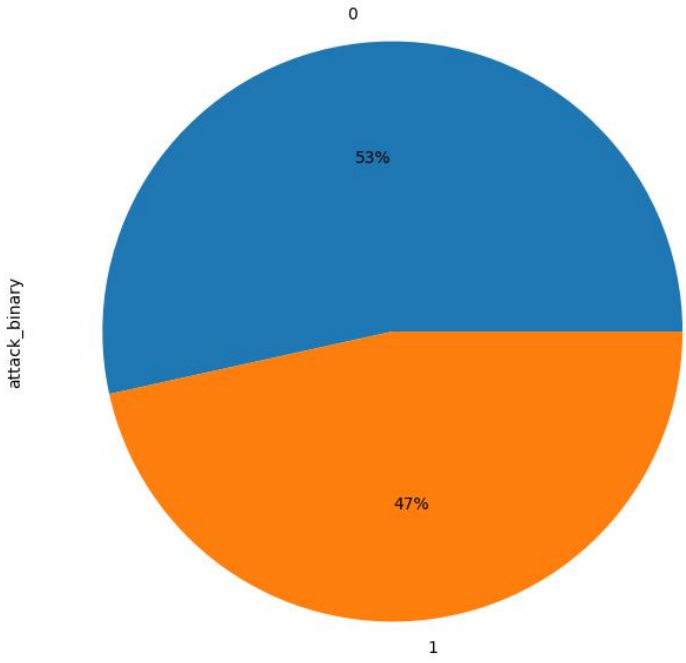


**Figure 5.** Plot of binary encoded data.

## 2.5. Building the model

After feature engineering, the hybrid decision tree model was built. The hybrid model developed was a three-fold classifier ensemble namely, a Classical Decision Tree-Light Gradient Boosting Machine-CatBoost hybrid model. The predictions from the Classical Decision Tree were fed as inputs into Light

Gradient Boosting Machine, while the predictions from Light Gradient Boosting Machine were fed into the CatBoost classifier in a stacking design.

### 2.5.1. Decision Tree classifier

The hyperparameters tuned for the decision tree model were as follows: max_features = 10, max_depth = 4, random_state = 1. The number of features were set at a maximum of ten so that the classifier would have enough features out of the entire feature set to make sufficient and accurate predictions. The random state was set to 1 to ensure that any time the model is run, the output results remain the same. The Decision Tree model was trained via the preprocessed NSL-KDD Train+ dataset and thereafter evaluated by relevant metrics. The splitting of the Decision Tree model was by Gini index as described in Equation (1)[18]:

$$\text{Gini index: } i(node) = 1 - \sum_{y \in y} p(y_i)^2 \tag{1}$$

where,

$i(node)$ = impurity of node $i$

$y_i$ = class (normal and attack)

$p(y_i)$ = probability of class occurrence

The respective probabilities of the two classes were obtained using Equations (2) and (3).

$$Probability\ of\ attack = \frac{Number\ of\ attack\ samples}{Number\ of\ normal\ samples + Number\ of\ attack\ samples} \tag{2}$$

$$Probability\ of\ normal = \frac{Number\ of\ normal\ samples}{Number\ of\ normal\ samples + Number\ of\ attack\ samples} \tag{3}$$

**Table 1** describes the components of the Decision Tree splits while **Figure 6** further displays the developed Decision Tree.

**Table 1.** Components of Decision Tree splits.

| X_features | Gini index | *N*_samples | *N*_values (Normal, Attack) |
|---|---|---|---|
| **x[12]** | 0.498 | 125,973 | 67,343, 58,630 |
| **x[8]** | 0.298 | 77,789 | 63,586, 14,203 |
| **x[15]** | 0.169 | 63,272 | 57,377, 5895 |
| **x[41]** | 0.489 | 11,403 | 6536, 4867 |
| | 0.067 | 4485 | 155, 4330 |
| | 0.143 | 6918 | 6381, 537 |
| **x[14]** | 0.039 | 51,869 | 50,841, 1028 |
| | 0.004 | 50,855 | 50,764, 91 |
| | 0.14 | 1014 | 77, 937 |
| **'x[33]** | 0.49 | 14,517 | 6209, 8308 |
| **'x[11]** | 0.419 | 6927 | 4860, 2067 |
| | 0.266 | 5757 | 4846, 911 |
| | 0.024 | 1170 | 14, 1156 |
| **'x[5]** | 0.292 | 7590 | 1349, 6241 |
| | 0.391 | 4607 | 1226, 3381 |

**Table 1**. (*Continued*).

| X_features | Gini index | *N*_samples | *N*_values (Normal, Attack) |
|---|---|---|---|
| | 0.079 | 2983 | 123, 2860 |
| **'x[43]** | 0.144 | 48,184 | 3757, 44,427 |
| **'x[21]** | 0.018 | 44,619 | 414, 44,205 |
| **'x[14]** | 0.011 | 44,431 | 244, 44,187 |
| | 0.01 | 44,404 | 219, 44,185 |
| | 0.137 | 27 | 25, 2 |
| **'x[38]** | 0.173 | 188 | 170, 18 |
| | 0.0 | 7 | 0, 7 |
| | 0.114 | 181 | 170, 11 |
| **'x[6]** | 0.117 | 3565 | 3343, 222 |
| **'x[38]** | 0.06 | 3418 | 3313, 105 |
| | 0.324 | 59 | 12, 47 |
| | 0.034 | 3359 | 3301, 58 |
| **'x[45]** | 0.325 | 147 | 30, 117 |
| | 0.053 | 73 | 2, 71 |
| | 0.47 | 74 | 28, 46 |

### 2.5.2. Light gradient boosting machine (LGBM) classifier

The hyperparameters for training the LGBM stage of the hybrid model were as follows: objective = 'binary', learning_rate = 0.05, n_estimators = 100, max_depth = 6 and random_state = 1. The 'objective' hyperparameter specifies the learning task and corresponding learning objective at hand for the algorithm. In this case it was specified as 'binary' since the problem being considered is a binary classification problem of '0' as normal traffic and '1' as attack traffic in the network. The Decision Tree predictions were added to the data features and the LGBM model was thereafter trained with the preprocessed NSL-KDD Train+ dataset, then evaluated.

The LGBM algorithm was based on a hundred Decision Trees hence the model formed is described below:

Given Training Data D (NSL-KDD Train+) = $\{\mathbf{x}_i, y_i{}_{i=1}^{m}\}$ ($\mathbf{x}_i \in \mathbb{R}^n, y_i \in$ Y) where 'm' is the samples and 'n' is features. In order to get the estimation by the LGBM classifier, the Decision Trees predictions are combined as follows:

$$\hat{y}_i^{LG} = \sum_{p}^{100} f_p(x_i) \tag{4}$$

where the number of trees is 100 and $f_p$ represents the Decision Tree prediction. The goal in this instance is minimization of the objective function in Equation (5) to get $f_p$.

$$f_p = \arg\min_{f_p} \sum_{i=1}^{m} L(y_i, \hat{y}_i^{LG(p)}) + \Omega(f_p) \tag{5}$$

where $L$ is the loss function and $\Omega$ is the regularization parameter. $\Omega$ is given by Equation (6).

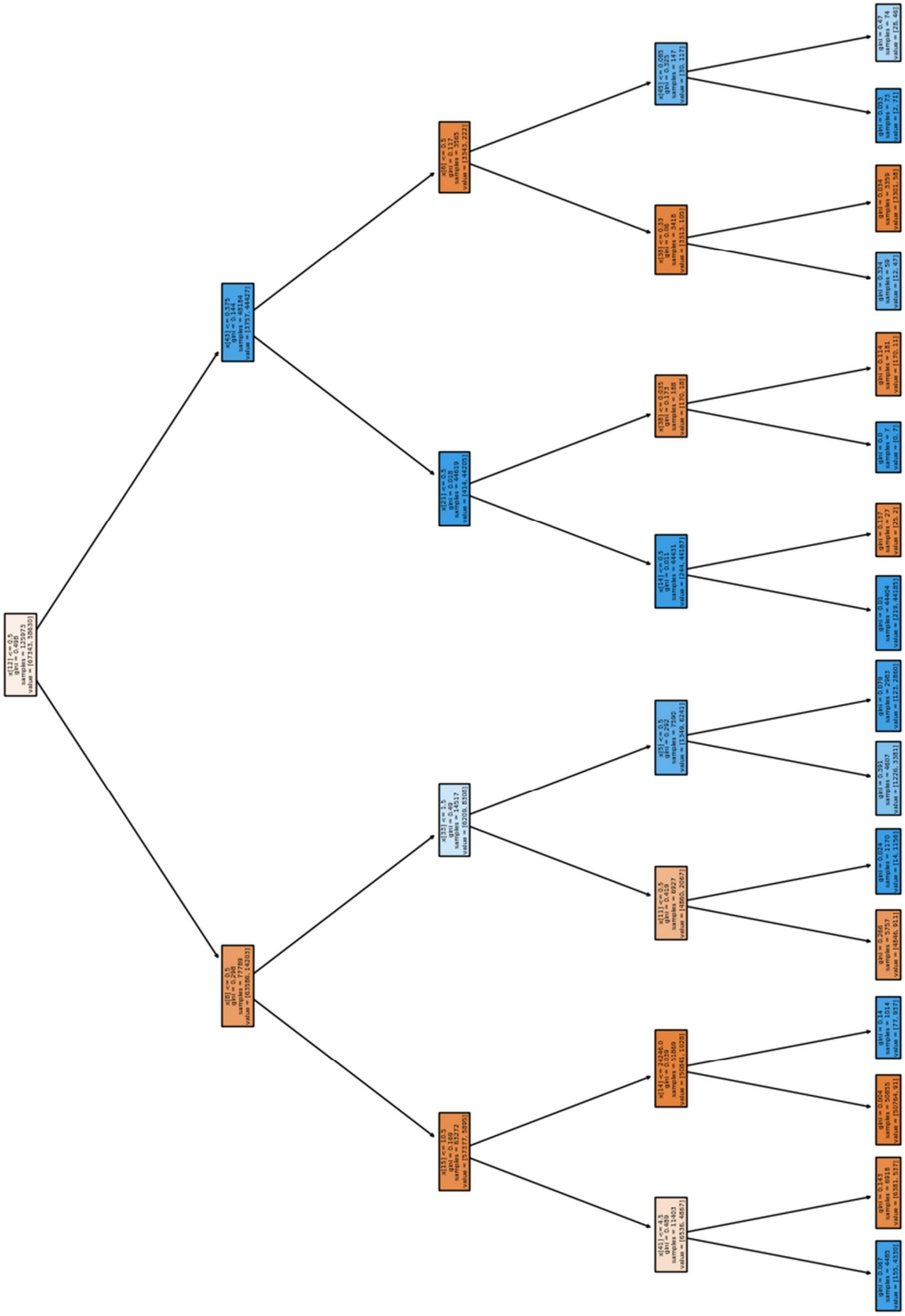

**Figure 6.** Decision Tree model.

$$\Omega(f_p) = \alpha T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \tag{6}$$

The arguments $\alpha$ and $\lambda$ are penalty parameters for '$T$' leaves and the weight of leaves '$w$', respectively. The loss function $L$ is a squared error, such that,

$$L(y_i, \hat{y}_i^{LG(p-1)} + f_p(x) = (y_i - \hat{y}_i^{LG(p-1)} - f_p(x)^2 = (r - f_p(x)^2 \tag{7}$$

The parameter '$r$' in Equation (7) is a residual that is fitted to get $f_p$. A quadratic approximation is used to define the function for minimization of the objective function at iteration $p$ as described in Equation (8).

$$\begin{aligned} f_p &\cong \arg\min_{f_p} \sum_{i=1}^{m} \left[ g_i f_p(x_i) + \frac{1}{2} h_i f_p^2(x_i) \right] + \Omega(f_p), \\ g_i &= \partial_{\hat{y}^{LG}(p-1)} L\left(y_i, \hat{y}_i^{LG(p-1)}\right), \\ h_i &= \partial^2_{\hat{y}^{LG}(p-1)} L\left(y_i, \hat{y}_i^{LG(p-1)}\right). \end{aligned} \tag{8}$$

Minimization of the objective function yields a new decision tree $f_p$. Each node that has the largest information gain is divided by the tree. The variance gain that belongs to a node which separates a feature '$j$' at point '$s$' is given by Equation (9).

$$Z_{j|O}(s) = \frac{1}{n_O} \left\{ \frac{\left(\sum_{\{x \in O: x_{ij} \le s\}} g_i\right)^2}{n^j_{l|O}(s)} + \frac{\left(\sum_{\{x \in O: x_{ij} > s\}} g_i\right)^2}{n^j_{r|O}(s)} \right\} \tag{9}$$

where,

$n_O$ is Decision Tree fixed node,

$O$ is samples on $n_O$,

$n^j_{l|O}(s) = \sum I[x \in O: x_{ij} \le s]$ representing the left node,

$n^j_{r|O}(s) = \sum I[x \in O: x_{ij} > s]$ representing the right node.

The decision tree therefore selects $s_j^* = arg \max_s Z_j(s)$ belonging to each feature '$j$' and proceeds to compute the greatest gain $Z_j(s_j^*)$. The data was thus split into both left and right nodes in accordance with the feature $j^*$ and point $s_{j^*}$. The entire data samples were scanned to locate the optimal point of split so as to deduce the information gain. The predictions of LGBM were fed into the final stage of the hybrid classifier model (i.e., CatBoost).

### 2.5.3. CatBoost classifier

The following hyperparameters were set: learning_rate = 0.05, verbose = 100, random_state = 1. The default number of decision trees for the classifier (i.e., 1000) was employed to develop the model. The verbose parameter was set to 100 so that the output can be displayed in steps of 100 as shown in **Figure 7**. The hybrid model was then trained and evaluated.

```
0:      learn: 0.5357541        total: 229ms    remaining: 3m 48s
100:    learn: 0.0089118        total: 14.4s    remaining: 2m 7s
200:    learn: 0.0066042        total: 27s      remaining: 1m 47s
300:    learn: 0.0052764        total: 39.6s    remaining: 1m 31s
400:    learn: 0.0051027        total: 47.5s    remaining: 1m 10s
500:    learn: 0.0051027        total: 57.9s    remaining: 57.6s
600:    learn: 0.0051027        total: 1m 6s    remaining: 44.2s
700:    learn: 0.0051027        total: 1m 13s   remaining: 31.5s
800:    learn: 0.0051027        total: 1m 22s   remaining: 20.5s
900:    learn: 0.0051026        total: 1m 29s   remaining: 9.88s
999:    learn: 0.0051026        total: 1m 38s   remaining: 0us

<catboost.core.CatBoostClassifier at 0x7f1c18fc9960>
```

**Figure 7.** Iterations for CatBoost Classifier.

The hybrid classifier was trained on all 125,973 samples as described in **Table 2** and tested on all 22,544 test samples as described in **Table 3**. Summary of hyperparameters tuned for the algorithms in Scikit-learn is described in **Table 4**.

**Table 2.** Hybrid classifier training parameters.

| **X_train** | | **Y_train** |
|---|---|---|
| ***N*_samples** | ***N*_features** | ***N*_samples** |
| 125,973 | 51 | 125,973 |

**Table 3.** Hybrid classifier test parameters.

| **X_test** | | **Y_test** |
|---|---|---|
| ***N*_samples** | ***N*_features** | ***N*_samples** |
| 22,544 | 51 | 22,544 |

**Table 4.** Hyperparameter values for classifier training.

| **Algorithm** | **Hyperparameters** | **Values** |
|---|---|---|
| Decision Tree | max_features | 10 |
| | max_depth | 4 |
| | random_state | 1 |
| LightGBoost | objective | 'binary' |
| | learning_rate | 0.05 |
| | n_estimators | 100 |
| | max_depth | 6 |
| | random_state | 1 |
| CatBoost | learning_rate | 0.05 |
| | verbose | 100 |
| | cat_features | cat_cols |
| | random_state | 1 |
| | iterations | Default (1000) |

The stacking procedure for obtaining the hybrid decision tree model is given in Appendix.

## 3. Results

The performances of the machine learning (ML) models were compared based on Accuracy, Recall, Precision, F1-Score, Receiver Operating Characteristic curve and Confusion Matrix. **Figures 8** to **10** are the confusion matrices for Decision Tree, LGBM and CatBoost classifiers respectively at final evaluation using the NSL-KDD test data.

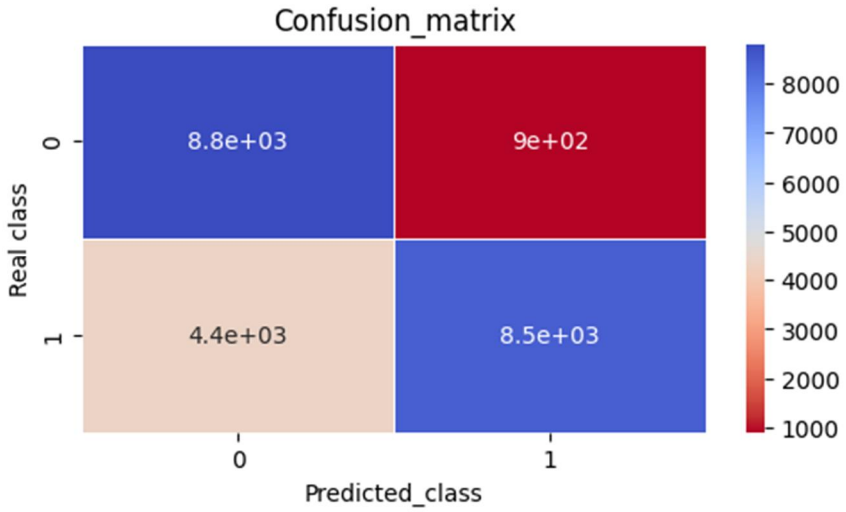


**Figure 8.** Decision Tree confusion matrix.

The values shown in the confusion matrix of **Figure 8** are True Negative (TN) = 8809, False Positive (FP) = 902, False Negative (FN) = 4371 and True Positive (TP) = 8462, respectively.

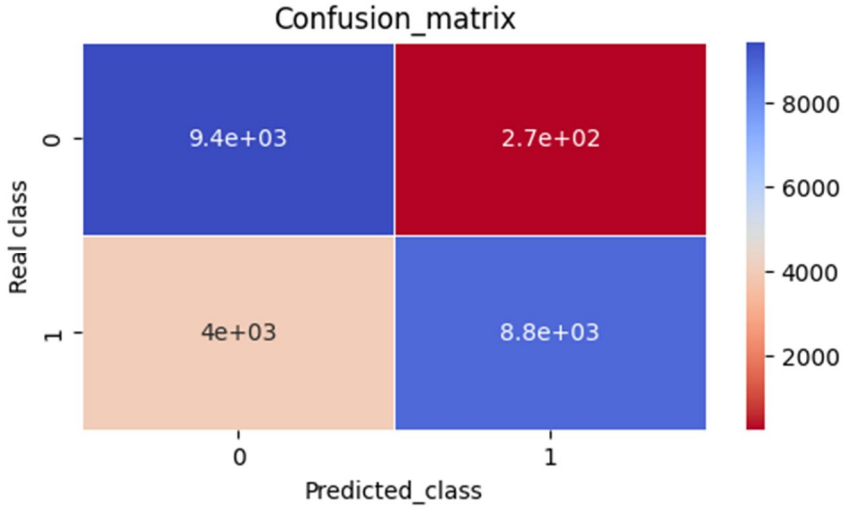


**Figure 9.** LGBM confusion matrix.

The values shown in the confusion matrix of **Figure 9** are True Negative (TN) = 9437, False Positive (FP) = 274, False Negative (FN) = 4024 and True Positive (TP) = 8809, respectively.

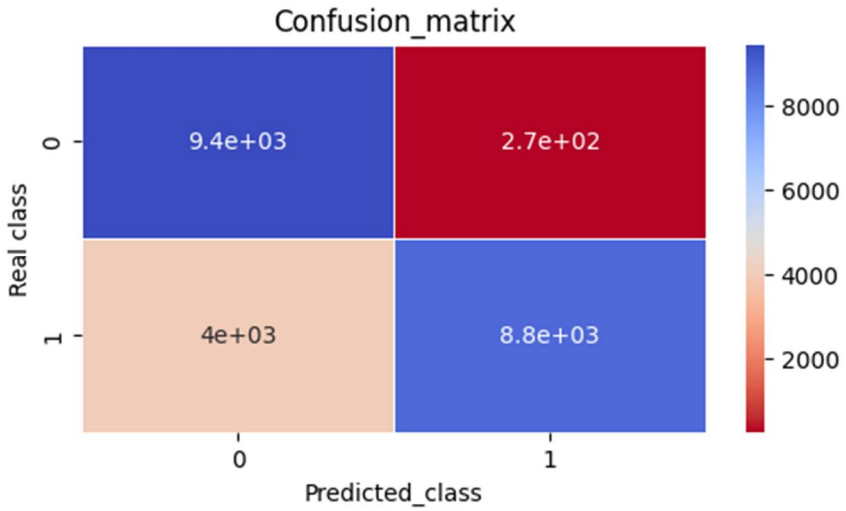


**Figure 10.** CatBoost confusion matrix.

The values shown in the confusion matrix of **Figure 10** are True Negative (TN) = 9437, False Positive (FP) = 274, False Negative (FN) = 4013 and True Positive (TP) = 8820, respectively.

From the confusion matrices, the accuracy, precision, recall and F1-Score metrices were obtained as described by Equation (10) to (13)[16].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{10}$$

$$Precision = \frac{TP}{TP + TN} \tag{11}$$

$$Recall = \frac{TP}{TP + FN} \tag{12}$$

$$F1 - score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{13}$$

Furthermore, the Receiver Operating Characteristic curves for each of the three classifiers at final evaluation are presented in **Figures 11** to **13**. The area under the curve (AUC) for the receiver operating characteristic curve is a measure of how the binary classifier distinguishes between classes[19]. The higher this value, the better the classifier can distinguish between positive and negative classes.

**Table 5** displays the summary of the performance results for all three classifier stages of the hybrid stacked model.

**Table 5.** Summary of test results for individual stages of hybrid model.

| Metric | Decision tree | Decision Tree-LGBM | Decision Tree-LGBM-CatBoost |
|---|---|---|---|
| **Precision** | 90% | 97% | 97% |
| **Recall** | 66% | 69% | 69% |
| **F1-score** | 76% | 80% | 80% |
| **Accuracy** | 77% | 81% | 81% |
| **False Negatives** | 4371 | 4024 | 4013 |
| **False Positives** | 902 | 274 | 274 |
| **Area Under Curve** | 0.94 | 0.93 | 0.96 |

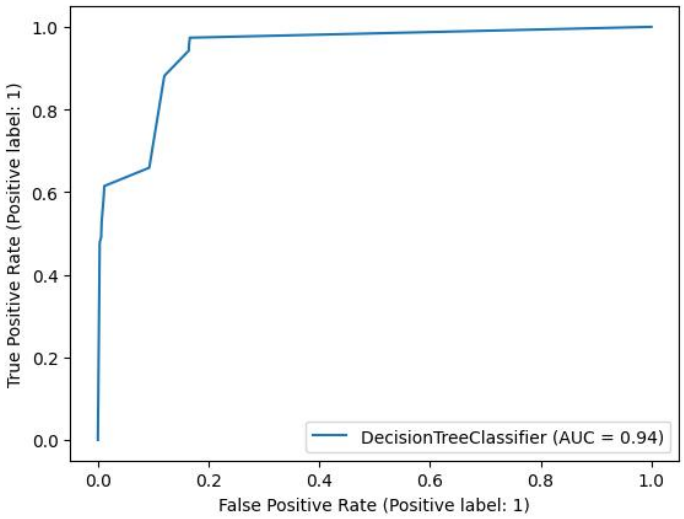


**Figure 11.** Decision Tree receiver operating characteristic curve.

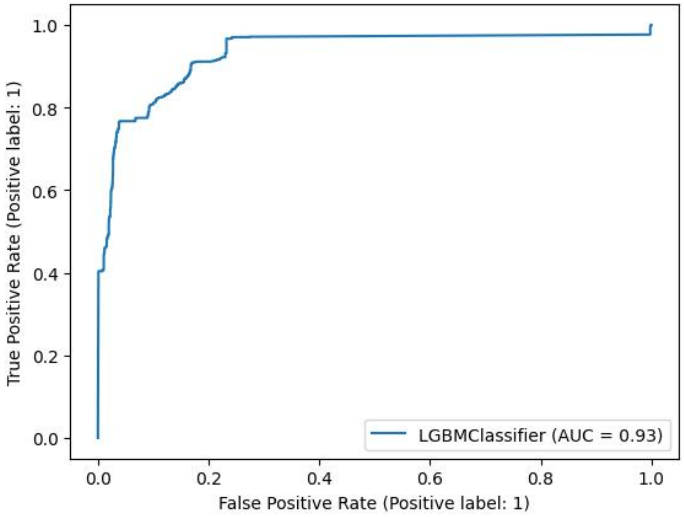


**Figure 12.** LGBM receiver operating characteristic curve.

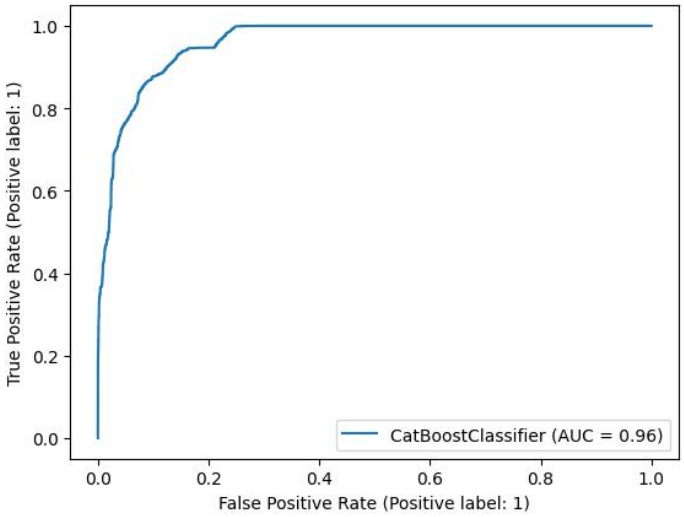


**Figure 13.** CatBoost receiver operating characteristic curve.

## 4. Discussion

This investigation was to determine the extent to which a hybrid variant of the Decision Tree algorithm would outperform the standard Decision Tree algorithm so as to develop a model capable of reducing the number of false positives in intrusion detection. The individual levels for the hybrid model were executed as follows. The NSL-KDD Train+ dataset was first fit with the Decision Tree algorithm and the predicted outcomes served as input to the LGBM classifier algorithm with the features of the NSL-KDD Train+ dataset. The predictions from the LGBM classifier further served as input for the CatBoost classifier, the final stage of the stacked model. The summary of results displayed in **Table 5** are discussed as percentage values in this section. The hybrid model with precision score, recall score, F1-score and accuracy scores of 97%, 69%, 80% and 81%, respectively was found to outperform the Classical Decision Tree model with corresponding values of 90%, 66%,76% and 77%, respectively. This outcome proved that hybridization of the Decision Tree classifier with boosted algorithms of LGBM and CatBoost leads to better performance results with the NSL-KDD dataset. Furthermore, the hybrid model presents a smaller number of False Positives and False Negatives which is desired in intrusion detection with values of 274 and 4013 respectively as against the Classical Decision Tree with values of 902 and 4371, respectively. Since a better intrusion detection model gives a reduced number of false predictions, the hybrid approach is proved to be better at detecting intrusions with the NSL-KDD dataset than the Classical Decision Tree model in this experiment. The area under curve (AUC) values displayed in **Table 5**, present the hybrid approach with 0.96 and the Classical Decision Tree with 0.94. An AUC value which is closer to 1 means a better performing model than one which is less close, the hybrid model on this basis is further proven to outperform the Classical Decision Tree model.

The results in this work were compared with the hybrid work by Alladi[15] in **Table 6**. The rationale for this comparison is that Alladi[15] also employed the NSL-KDD dataset as a benchmark. The percentage accuracy scores are compared below.

**Table 6.** Results comparison.

| Author | Classifier | Accuracy % |
|---|---|---|
| Alladi[15] | MLP & k-means clustering | 73.09 |
| Proposed hybrid model | Decision Tree, LGBM and CatBoost | 81 |

As displayed in **Table 6**, the hybridization approach employed in this work outperforms the approach by Alladi[15]. Other works studied in literature were also surpassed in terms of accuracy.

## 5. Conclusion

This work involved the development of a hybrid model based on Decision Trees. Stacking approach was adopted where the predictions from one classifier were fed into another so as to improve performance. Three classifiers were adopted in developing the model namely, Standard or Classical Decision Tree, Light Gradient Boosting Machine and Catboost. The predictions from Decision Tree on the NSL-KDD dataset served as input for the LGBM algorithm while the predictions from the LGBM classifier algorithm were fed into the CatBoost classifier to form the final level of the stacked model. Results prove that the hybrid model had improved performance compared to the Standard Decision Tree in terms of precision, accuracy, recall and F1-score. The AUC value of the hybrid model was also an improvement as well as number of False Negatives and False Positives.

## Author contributions

Conceptualization, EO and PO; methodology, EO and PO; software, EO; validation, EO, PO and IU; formal analysis, IU; investigation, EO; resources, PO; data curation, EO; writing—original draft preparation, EO; writing—review and editing, PO and IU; visualization, IU; supervision, PO; project administration, PO; funding acquisition, PO and IU. All authors have read and agreed to the published version of the manuscript.

## Conflict of interest

The authors declare no conflict of interest.

## References


1. Mohajan HK. Third industrial revolution brings global development. *Journal of Social Sciences and Humanities* 2021; 7(4): 239–251.
2. Orukpe PE, Erhiaguna TO, Agbontaen FO. Computer security and privacy in wireless local area network in Nigeria. *International Journal of Engineering Research in Africa* 2013; 9: 23–33. doi: 10.4028/www.scientific.net/jera.9.23
3. Shruti M. Types of cyber attacks you should be aware of in 2023. Available online: https://www.simplilearn.com/tutorials/cyber-security-tutorial/types-of-cyber-attacks (accessed on 11 August 2023).
4. FORTINET. Types of cyber attacks. Available online: https://www.fortinet.com/resources/cyberglossary/types-of-cyber-attacks (accessed on 11 August 2023).
5. DNV. The three-pillar approach to cyber security: Data and information protection. Available online: https://www.dnv.com/article/the-three-pillar-approach-to-cyber-security-data-and-information-protection-165683 (accessed on 11 August 2023).
6. James K. What are the 5 pillars of cybersecurity? Available online: https://cybersecurityforme.com/what-are-the-5-pillars-of-cybersecurity (accessed on 11 August 2023).
7. Osa E. Cyber security terminologies and concepts. In: *SMART-IEEE-ACity-ICTU-CRACC-ICTU-Foundations Series Book Chapter on Web of Deceit.* Creative Research Publishers; 2022. pp. 231–236.
8. Rama Devi R, Abualkibash M. Intrusion detection system classification using different machine learning algorithms on KDD-99 and NSL-KDD datasets—A review paper. *International Journal of Computer Science and Information Technology* 2019; 11(3): 65–80. doi: 10.5121/ijcsit.2019.11306
9. Abuh A, Orukpe PE. Development of an integrated campus security alerting and access control system. In: *Emerging Trends in Engineering Research and Technology Vol. 9*. Book Publisher International; 2020. pp. 57–67.
10. Chudasma P. *Network Intrusion Detection System using Classification Techniques in Machine Learning* [Master's thesis]. Dublin Business School; 2020.
11. Sangfor Technologies. What is an intrusion detection system and how does it work. Available online: https://www.sangfor.com/glossary/cybersecurity/what-is-intrusion-detection-system-and-how-does-it-work (accessed on 11 August 2023).
12. Velimirovic A. What is an intrusion detection system? Available online: https://phoenixnap.com/blog/intrusion-detection-system (accessed on 11 August 2023).
13. FORTINET. Intrusion Detection System (IDS). Available online: https://www.fortinet.com/resources/cyberglossary/intrusion-detection-system (accessed on 11 August 2023).
14. Géron A. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, Inc.; 2019.
15. Alladi SK. *Effectively Improving the Efficiency and Performance of an Intrusion Detection System Using Hybrid Machine Learning Models* [Master's thesis]. National College of Ireland; 2020.
16. Sapre S, Ahmadi P, Islam K. A robust comparison of the KDDCup99 and NSL-KDD IoT network intrusion detection datasets through various machine learning algorithms. arXiv 2019; arXiv:1912.13204. doi: 10.48550/arXiv.1912.13204
17. Vinayakumar R, Alazab M, Soman KP, et al. Deep learning approach for intelligent intrusion detection system. *IEEE Access* 2019; 7: 41525–41550. doi: 10.1109/access.2019.2895334
18. Li Y, Gao J, Li Q, Fan W. Ensemble learning. In: *Data Classification Algorithms and Applications' Data Mining and Knowledge Discovery Seri*es. CRC Press; 2015. pp. 498–500.
19. Bhandari A. Guide to AUC ROC Curve in machine learning: What is specificity? Available online: www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/ (accessed on 8 June 2023).

# Appendix

**Algorithm for Implementation of Hybrid Stacking**

**Input:** Training Data D (NSL-KDD Train+) = $\{\mathbf{x}_i, y_i{}_{i=1}^{m}\}$ ($\mathbf{x}_i \in \mathbb{R}^n, y_i \in$ Y)

**Output: A hybrid ensemble classifier *H***

1: Step 1: Learn first-level classifiers ($h_1$, $h_2$)

2: **for** $t \leftarrow$1 to 2 **do**

3: Learn a classifier $h_t$ from D

4: **end for**

5: Step 2: Construct new data sets from D

6: **for** $i \leftarrow$1 to $m$ **do**

7: Construct a new data set that contains $\{\mathbf{x}'_i, y_i\}$, where $\mathbf{x}'_i = \{h_1(\mathbf{x}_i), h_2(\mathbf{x}_i)\}$

8: **end for**

9: Step 3: Learn a final-level classifier

10: Learn a new classifier $h'$ based on the newly constructed data set

11: **return** $H(\mathbf{x}) = h'(h_1(\mathbf{x}), h_2(\mathbf{x}))$