

# Random Forest Hyperparameter Tuning in Machine Learning for Improved Performance in Intrusion Detection Systems

Dr. Amaku Amaku<sup>1</sup>, Dr. Igbinosa O. G<sup>2</sup>, Dr. Kingsley Okorie<sup>3</sup>, Amodu Felix<sup>4</sup>

*1Department of Computer Science, Veritas University, Abuja, Nigeria.*

*2Department of Electrical/Electronic and Telecoms Engineering, Bells University, Nigeria.*

*3Department of Computer Science, Enugu State University of Science and Technology, Nigeria.*

*42Department of Electrical/Electronic and Telecoms Engineering, Bells University, Nigeria.*

Date of Submission: 05-01-2025

Date of Acceptance: 15-01-2025

## ABSTRACT

In recent times, researchers have been proposing different Intrusion Detection methods to deal with the increasing number and complexity of threats as technology keeps emerging. In this context, Random Forest models have been providing a notable performance on her predictive capacity to applications in the realm of behavioural-based Intrusion Detection Systems and other related fields of specialization which includes medicines, Banking, commerce, etc in terms high magnitude forecasting and optimal predictions. In this work, in-depth evaluation analysis of the Random Forest tuning were carried out with respect to classification, feature selection, and proximity metrics. This empirical research will provide an inclusive review of the general basic concepts related to Intrusion Detection Systems, which includes taxonomies, data collection, modeling and evaluation metrics. Furthermore, the manual hyperparameter tuning technique was used for this research work and a desirable experimental output was achieved as showed in this work.

Key Words: Random Forest, Machine Learning, Optimization, Hyperparameters, Classification, Evaluation Metrics.

## I. INTRODUCTION

### MACHINE LEARNING

Machine learning (ML) algorithms have been widely used in many applications domains, including advertising, recommendation systems, computervision, natural language processing, and user behavior analytics (Jordan & Mitchell,2015).

This is because they are generic and demonstrate high performance in data analytics problems. Different ML algorithms are suitable for different types of problems or datasets (Ziler& Huber,2019). In general, building an effective machine learning model is a complex and time-consuming process that involves determining the appropriate algorithm and obtaining an optimal model architecture by tuning its hyper-parameters (HPs) (Shawi ,et al,2019). Two types of parameters exist in machine learning models: one that can be initialized and updated through the data learning process (e.g., the weights of neurons in neural networks), named model parameters; while the other, named hyper-parameters, cannot be directly estimated from data learning and must be set before training a ML model because they define the architecture of a ML model (Kuhn & John, 2013). Hyper-parameters are the parameters that are used to either configure a ML model (e.g., the penalty parameter C in a support vector machine, and the learning rate to train a neural network) or to specify the algorithm used to minimize the loss function (e.g., the activation function and optimizer types in a neural network, and the kernel type in a support vector machine) (Diaz et al, 2017). To build an optimal ML model, a range of possibilities must be explored. The process of designing the ideal model architecture with an optimal hyper-parameter configuration is named hyper-parameter tuning. Tuning hyper-parameters is considered a key component of building an effective ML model, especially for tree-based ML models and deep neural networks, which have many hyper-

parameters (Hutter et al,2019). Hyper-parameter tuning process is different among different ML algorithms due to their different types of hyper-parameters, including categorical, discrete, and continuous hyper-parameters (Decastro-Garca et al,2019). Manual testing is a traditional way to tune hyper-parameters and is still prevalent in graduate student research, although it requires a deep understanding of the used ML algorithms and their hyper-parameter value settings (Abreu, 2019). However, manual tuning is ineffective for many problems due to certain factors, including a large number of hyper-parameters, complex models, time consuming model evaluations, and non-linear hyper-parameter interactions. These factors have inspired increased research in techniques for automatic optimization of hyper-parameters; so-called hyper-parameter optimization. (HPO) (Steinholtz,2018). The main aim of HPO is to automate hyper-parameter tuning process and make it possible for users to apply machine learning models to practical problems effectively (Shawi ,et al,2019). The optimal model architecture of a ML model is expected to be obtained after a HPO process. Some important reasons for applying HPO techniques to ML models are as follows (Hutter et al,2019):

1. It reduces the human effort required, since many ML developers spend considerable time tuning the hyper-parameters, especially for large datasets or complex ML algorithms with a large number of hyper-parameters.
2. It improves the performance of ML models. Many ML hyper-parameters have different optimums to achieve best performance in different datasets or problems.
3. It makes the models and research more reproducible. Only when the same level of hyper-parameter tuning process is implemented can different ML algorithms be compared fairly; hence, using a same HPO method on different ML algorithms also helps to determine the most suitable ML model for a specific problem. It is crucial to select an appropriate optimization technique to detect optimal hyper-parameters. Traditional optimization techniques may be unsuitable for HPO problems, since many HPO problems are non-convex or non-differentiable optimization problems, and may result in a local instead of a global optimum (Lou, 2016). Gradient descent-based methods are a common type of traditional optimization algorithm that can be used to tune continuous hyper-parameters by calculating their gradients (Maclaurin et al, 2015). For

example, the learning rate in a neural network can be optimized by a gradient-based method. Compared with traditional optimization methods like gradient descent, many other optimization techniques are more suitable for HPO problems, including decision-theoretic approaches, Bayesian optimization models, multifidelity optimization techniques, and metaheuristics algorithms (Decastro-Garca et al,2019). Apart from detecting continuous hyper-parameters, many of these algorithms also have the capacity to effectively identify discrete, categorical, and conditional hyper-parameters. Decision-theoretic methods are based on the concept of defining a hyper-parameter search space and then detecting the hyper-parameter combinations in the search space, ultimately selecting the best-performing hyper-parameter combination.

Bergstra et al, 2019 concluded that Grid search (GS) is a decision-theoretic approach that involves exhaustively searching for a fixed domain of hyper-parameter values. . James & Yoshua,2019 also discussed Random search (RS) as another decision-theoretic method that randomly selects hyper-parameter combinations in the search space, given limited execution time and resources. In GS and RS, each hyper-parameter configuration is treated independently.

## 1.2 DECISION TREE

Decision Tree is a graphical representation of all possible solutions to a decision, decision tree is based on some conditions and it can be easily be explained. It represents a function that takes as Input a vector of attribute values and returns a “decision” – a single output value.

Decision tree is a flow-chart-like tree structure that uses a branching method to illustrate every possible outcome of a decision. Each node within the tree represents a test on a specific variable- and each branch is the outcome of that test. It is also a simple flowchart that selects labels for input values.

This flowchart consists of decision nodes, which check feature values, and leaf nodes, which assign labels. To choose the label for an input value, we begin at the flowchart’s initial decision nodes, known as its roots node. This node contains a condition that checks one of the input value’s features, and selects a branch based on that features value. Following the branch that describes our input value, we arrive at a new decision node, with a new condition on the input value’s features. We continue following the branch selected by each

node's condition, until we arrive at a leaf node which provides a label for the input value.

Decision tree algorithm falls under the category of supervise learning. They can be used to  
**For Example**

solve both regression and classification problems. A decision tree reaches its decision by performing a sequence of tests.

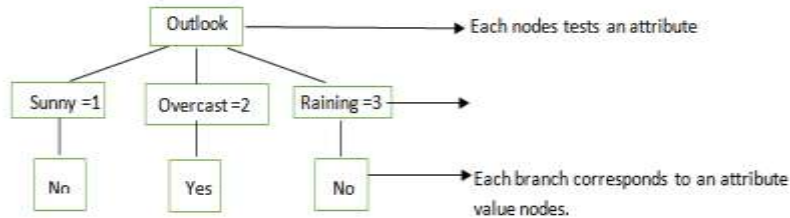


Figure 1.1 Decision Tree Learning Algorithm

**ID<sub>3</sub> ( Iterative Dichotomies 3)**

- ID<sub>3</sub> is on of the most common decision tree algorithm.
- Dichotomies means dividing into two completely opposite things.
- Algorithm iterative divides attribute into two groups are the most dominant attribute and others to construct a tree.
- Then, it calculate the Entropy and information gain of each attribute. In this way, the most dominant attribute can be founded.
- After then, the most dominant one is put on the tree as decision node. For
- Entropy and gain scores would be calculated again among the other attributes.
- Procedure continues until reaching a decision for that branch.

**Formulas:**

$$\text{Entropy}(s) = \epsilon - P(I) \cdot \text{Log}P_2(I)$$

..... (1)

$$\text{Gain}(S,A) = \text{Entropy}(s) - \epsilon [P(S/A) \cdot \text{Entropy}(S/A)]$$

..... (2)

A decision tree is also a simple flowchart that selects labels for input values. This flowchart consists of decision nodes, which check feature values, and leaf nodes, which assign labels. To choose the label for an input value, we begin at the flowchart's initial decision nodes, known as its roots node. This node contains a condition that checks one of the input value's features, and selects a branch based on that features value. Following the branch that describes our input value, we arrive at a new decision node, with a new condition on the input value's features. We continue following the branch selected by each node's condition, until we arrive at a leaf node which provides a label for the input value.

Once we have a decision tree, it is straightforward to use it to assign labels to new input values. What's less straightforward is how we can build a decision tree that models a given training set. But before we look at the learning algorithm for building decision tress, we'll consider a simpler task: picking the best "decision stump" for a **corpus**.

**A decision stump** is a decision tree with a single node that decides how to classify inputs based on a single feature. It contains one leaf for each possible feature value, specifying the class label that should be assigned to inputs whose features have that value. In order to build a decision stump, we must first decide which features should be used. The simplest method is to just build a decision stump for each possible feature, and see which one achieves the highest accuracy on the training data, although there are other alternatives that we will discuss later. Once we've picked a feature, we can build the decision stump by assigning a label to each based on the most frequently for the selected examples in the training set (i.e. the examples where the selected feature has that value).

Given the algorithm for choosing decision stumps, the algorithm for growing larger decision tress is straightforward. We begin by selecting the overall best decision stump for the classification task. We then check the accuracy of each of the leaves on the training set. Leaves that do not achieve sufficient accuracy are then replaced by new decision stumps, trained on the subset of the training corpus that is selected by the path to the leaf.

**1.3 Entropy and information Gain**

There are several methods for identifying the most informative feature for a decision stump.

One popular alternative called **information gain**, measures how much more organized the input values become when we divide them up using a given feature. How disorganized the original set of input values are, we calculate entropy of their labels, which will be high if the input values have highly varied labels, and how if many input values all have the same label. In particular, **entropy** is defined as the sum of the probability of each label times the log probability of that same label:

$$H = \sum_{l \in \text{labels}} P(l) * \log_2 P(l) \dots \dots \dots (3)$$

For example, Figure above shows how the entropy of labels in the weather prediction task depends on the ratio of sunny to outcast to raining attributes names. Note that if

Most input values have the same label (e.g., if P(sunny) is near 0 or near 1), then entropy is low. In particular, labels that have low frequency do not contribute much to the entropy (since P(l) is small), and labels with high frequency also do not contribute much to the entropy (since log<sub>2</sub>P(l) is small). On the other hand, if the input values have a wide variety of labels, then there are many labels with a “medium” frequency, where neither P(l) nor log<sub>2</sub>P(l) is small, so the entropy is high.

Once we have calculated the entropy of the label of the original set of input values, we can determine how much more organized the labels become once we apply the decision stump. To do so, we calculate the entropy for each of the decision stump’s leaves, and take the average of those leaf entropy values (weighed by the number of samples in each leaf). The information gain is then equal to the original entropy minus this new reduced entropy. The higher the information gain, the better job the decision stump does of dividing the input values into coherent groups, so we can build decision trees by selecting the decision stumps with the highest information gain.

Another consideration for decision tree is efficiency. The simple algorithm for selecting decision stumps described earlier must construct a weather decision stump for every possible feature, and this process must be repeated for every node in the constructed decision tree. A number of algorithms have been developed to cut down on the training time by storing and reusing information about previously evaluated examples.

However, decision trees also has a few disadvantages. One problem is that, since each branch in the decision tree splits the training data, the amount of training data available to train nodes

lower in the tree can become quite small. As a result, these lower decision nodes may overfit the training set, learning patterns that reflect idiosyncrasies of the training set rather than linguistically significant patterns in the underlying problem. One solution to this problem is to stop diving nodes once the amount of training data becomes too small. Another solution is to grow a full decision tree, but then to **prune** decision nodes that do not improve performance on a dev-test.

A second problem with decision trees is that they force features to be checked in a specific order, even when features may act relatively independently of one another. For example, when classifying documents into topics (such as a sports, automotive, or murder mystery), features such as has word (football) are highly indicating of a specific label, regardless of what the other feature value are. Since there is limited space near the top of the decision tree, most of these features will need to be repeated on many different branches in the tree. And since the number of branches increases exponentially as we go down the tree, the amount of repetition can be very large.

A related problem is the decision trees are not good at making use of features that re weak predictors of the correct label. Since these features make relatively small incremental improvements, they tend to occur very low in the decision tree. But by the time the decision tree learner has descended far enough to use these features, there is not enough training data left to reliable determine what effect they should have. If we could instead look at the effect of these features across the entire training set, then we might be able to make some conclusions about how they should affect the choice of label.

The face that decision trees require that features be checked in a specific order limits their ability to exploit features that are relatively independent of one another.

Computer security is the ability to protect a computer system and its resources in reference to Confidentiality, Integrity and Availability (Urasva, 2015). The main goal of any Intrusion Detection System is to detect attacks. Random forests (Breiman, 2001) are considered as one of the most successful general-purpose algorithms in modern-times (Biau and Scornet, 2016). They can be applied to a wide range of learning tasks, but most prominently to classification and regression. A random forest is an ensemble of trees, where the construction of each tree is random. After building an ensemble of trees, the random forest makes predictions by averaging the predictions of

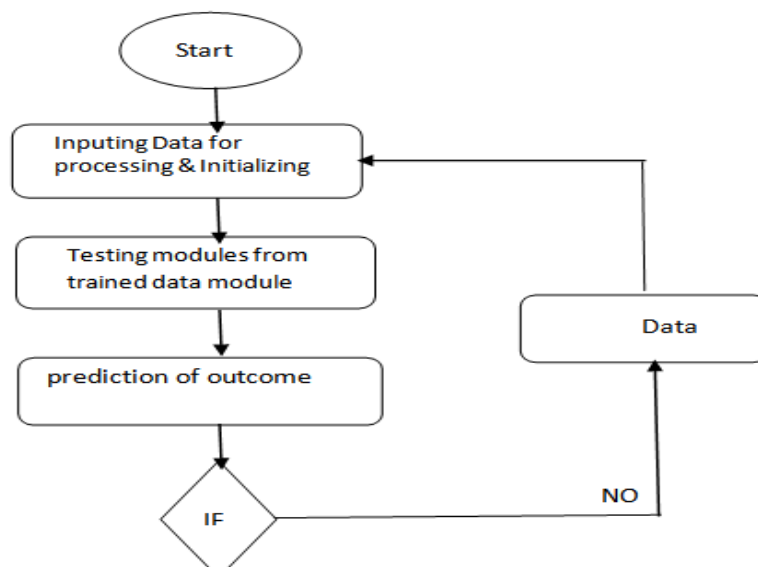
individual trees. Random forests often make accurate and robust predictions, even for very high-dimensional problems (Biau, 2012), in a variety of applications (Criminisi and Shotton, 2013; Belgiu and Dragut, 2016; D'iaz-Uriarte and Alvarez de Andrés, 2006). Recent theoretical works have established a series of consistency results of different variants of random forests, when the forests' parameters are tuned in certain ways (Scornet, 2016; Scornet et al., 2015; Biau, 2012; Biau et al., 2008). Random forest has nearly the same hyperparameters as a decision tree or a bagging classifier. Furthermore, knowing the attacks and how they are classified is important to enable better comprehension and critical analysis. In Table 1 represent the first dimension of the attack taxonomy proposed by Hansman and Hunt (2005), which is used in other works (e.g., Bhuyan et al. (2014), Ghorbani et al. (2010), and Sperotto et al. (2010)) and provides a good understanding about attacks on networks.

In the literature, there are a variety of attack taxonomies devoted to specific major attack types or attacked systems or protocols, such as DDoS attacks (Mirkovic and Reiher 2004), cloud systems (Juliadotter and Choo 2015; Mishra et al. 2017), web applications (Alvarez and Petrovic 2003; Watson 2007), Supervisory Control and Data Acquisition (SCADA) systems (Zhu et al. 2011), protocol DNP3 that is usually used in SCADA systems (East et al. 2009), P2P communication (Yue et al. 2009), embedded systems (Papp et al. 2015), botnets (Dagon et al. 2007), and the Internet infrastructure, for example, attacks on the Border Gateway Protocol (BGP) used for routing in the

Internet (Chakrabarti and Manimaran 2002). Taxonomies also can be oriented to attack response (Souissi and Serhrouchni 2014; Wu et al. 2011), target systems, causes, impact, time, among other characteristics. In 2008, Ijure and Williams had explored on classical attacks on taxonomies.

### RESEARCH DESIGN

This research work deals with an optimization of behavioral based random forest algorithm as a machine learning tool in intrusion detection systems. The research methodology will effectively discuss possible directions which this research work will take in order to achieve its objectives. The detailed information on the process involved in data acquisition will be presented also. Hyperparameters are important for machine learning algorithms since they directly control the behaviors of training algorithms and have a significant effect on the performance of machine learning models. Several techniques have been developed and successfully applied for certain application domains. This research work has proposed a technique which will improve a dataset's data content by translating it into a brand new feature subspace of lower dimensionality than the original. Normally machine learning algorithm transforms a problem that needs to be solved into an optimization problem and uses different optimization methods to solve the problem. The optimization function is composed of multiple hyperparameters that are set prior to the learning process and affect algorithm performance of the model.



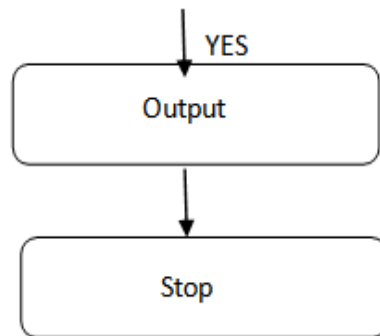
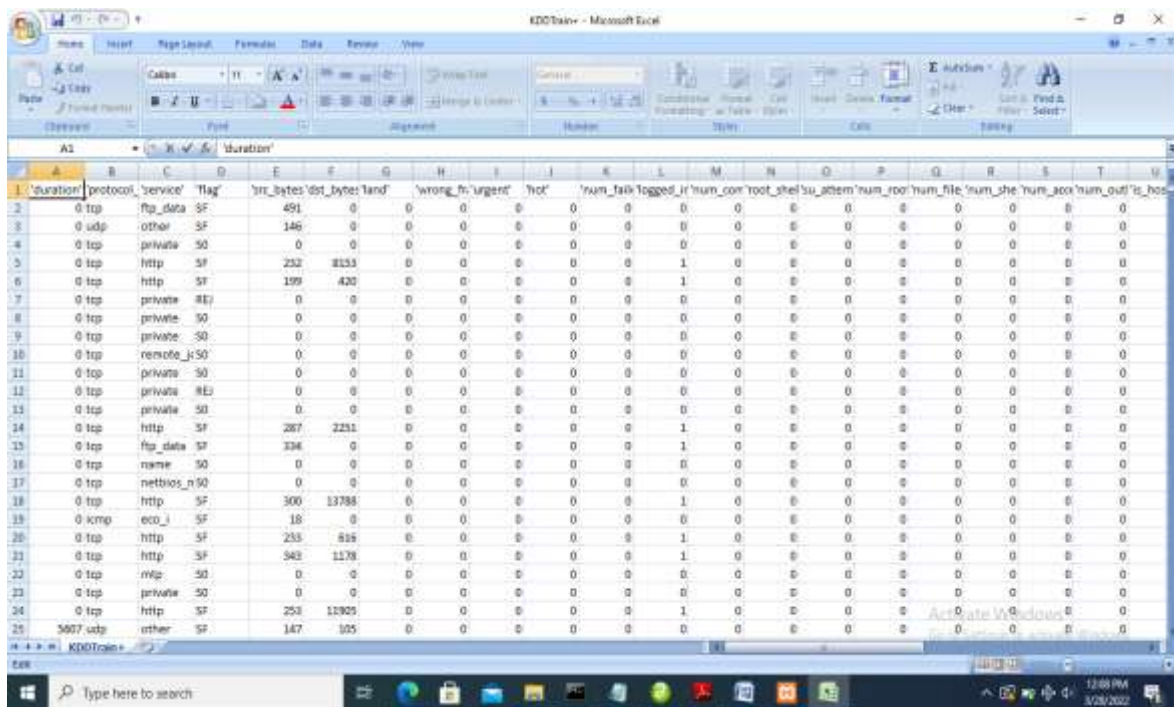


Fig 3.2 Program Flowchart



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	
1	'duration'	'protocol_type'	'service'	'flag'	'src_bytes'	'dst_bytes'	'land'	'wrong_fr'	'urgent'	'hot'	'num_failed_logins'	'logged_in'	'num_conn'	'root_shell'	'su_attempt'	'num_rpo'	'num_file'	'num_sh'	'num_acc'	'num_out'	'is_hos'
2	0	tcp	ftp_data	SF	491	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	udp	other	SF	146	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	tcp	http	SF	232	1153	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
6	0	tcp	http	SF	199	420	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
7	0	tcp	private	REI	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	tcp	remote_js	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	tcp	private	REI	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	tcp	http	SF	287	2251	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
15	0	tcp	ftp_data	SF	334	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
16	0	tcp	name	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	tcp	netbios_n	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	tcp	http	SF	300	13788	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
19	0	icmp	eco_i	SF	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	tcp	http	SF	233	819	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
21	0	tcp	http	SF	343	1178	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
22	0	tcp	ntp	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	tcp	private	S0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	0	tcp	http	SF	253	11905	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
25	3407	udp	other	SF	147	505	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Fig 3.3 dataset in csv(comer separated values) format file

The Info on dataset used is given below:

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 125973 entries, 0 to 125972
Data columns (total 42 columns):
# Column          Non-Null Count  Dtype
---  ---          ---
0 'duration'       125973 non-null int64
1 'protocol_type' 125973 non-null object
2 'service'       125973 non-null object
3 'flag'          125973 non-null object
4 'src_bytes'     125973 non-null int64
5 'dst_bytes'     125973 non-null int64
6 'land'         125973 non-null int64
7 'wrong_fragment' 125973 non-null int64
8 'urgent'        125973 non-null int64
9 'hot'          125973 non-null int64
10 'num_failed_logins' 125973 non-null int64
    
```

```

11 'logged_in'                125973 non-null int64
12 'num_compromised'         125973 non-null int64
13 'root_shell'              125973 non-null int64
14 'su_attempted'           125973 non-null int64
15 'num_root'                125973 non-null int64
16 'num_file_creations'     125973 non-null int64
17 'num_shells'              125973 non-null int64
18 'num_access_files'       125973 non-null int64
19 'num_outbound_cmds'      125973 non-null int64
20 'is_host_login'          125973 non-null int64
21 'is_guest_login'         125973 non-null int64
22 'count'                   125973 non-null int64
23 'srv_count'               125973 non-null int64
24 'serror_rate'             125973 non-null float64
25 'srv_serror_rate'        125973 non-null float64
26 'rerror_rate'            125973 non-null float64
27 'srv_rerror_rate'        125973 non-null float64
28 'same_srv_rate'          125973 non-null float64
29 'diff_srv_rate'          125973 non-null float64
30 'srv_diff_host_rate'     125973 non-null float64
31 'dst_host_count'         125973 non-null int64
32 'dst_host_srv_count'     125973 non-null int64
33 'dst_host_same_srv_rate' 125973 non-null float64
34 'dst_host_diff_srv_rate' 125973 non-null float64
35 'dst_host_same_src_port_rate' 125973 non-null float64
36 'dst_host_srv_diff_host_rate' 125973 non-null float64
37 'dst_host_serror_rate'   125973 non-null float64
38 'dst_host_srv_serror_rate' 125973 non-null float64
39 'dst_host_rerror_rate'   125973 non-null float64
40 'dst_host_srv_rerror_rate' 125973 non-null float64
41 'class'                   125973 non-null object
  
```

dtypes: float64(15), int64(23), object(4)

memory usage: 40.4+ MB

Following the collection of data obtained, the data collected was checked for the presence of error in data entry including misspellings and

missing data. Following this process, there was no error in misspelling of any data in the record.

HANDLING MISSING VALUE (This is to check which feature contains missing values)

```

print(df.isnull().sum())
'duration'                0
'protocol_type'           0
'service'                  0
'flag'                     0
'src_bytes'                0
'dst_bytes'                0
'land'                     0
'wrong_fragment'          0
'urgent'                   0
'hot'                      0
'num_failed_logins'        0
'logged_in'                0
'num_compromised'         0
'root_shell'              0
'su_attempted'            0
  
```

```
'num_root'          0
'num_file_creations' 0
'num_shells'        0
'num_access_files'  0
'num_outbound_cmds' 0
'is_host_login'     0
'is_guest_login'    0
'count'             0
'srv_count'         0
'serror_rate'       0
'srv_serror_rate'   0
'rerror_rate'       0
'srv_rerror_rate'   0
'same_srv_rate'     0
'diff_srv_rate'     0
'srv_diff_host_rate' 0
'dst_host_count'    0
'dst_host_srv_count' 0
'dst_host_same_srv_rate' 0
'dst_host_diff_srv_rate' 0
'dst_host_same_src_port_rate' 0
'dst_host_srv_diff_host_rate' 0
'dst_host_serror_rate' 0
'dst_host_srv_serror_rate' 0
'dst_host_rerror_rate' 0
'dst_host_srv_rerror_rate' 0
'class'             0
dtype: int64
```

### 7 Evaluation Metrics

The evaluation metrics generated from this research work is given below;  
 Parameter distribution of random forest used for the randomized search

# Number of trees to use for building the random forest

```
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 80, num = 10)]
```

# Number of features to consider at every split

```
max_features = ['auto', 'sqrt']
```

# Maximum number of levels in tree

```
max_depth = [2,4]
```

# Minimum number of samples required to split a node

```
min_samples_split = [2, 5]
```

# Minimum number of samples required at each leaf node

```
min_samples_leaf = [1, 2]
```

```
criterion = ['gini', 'entropy']
```

# Method of selecting samples for training each tree

```
bootstrap = [True, False]
```

Parameter distribution code

# Create the param grid

```
param_grid = {'n_estimators': n_estimators,
              'max_features': max_features,
              'max_depth': max_depth,
              'min_samples_split': min_samples_split,
              'min_samples_leaf': min_samples_leaf,
              'criterion': criterion,
              'bootstrap': bootstrap}
```

```
print(param_grid)
```

OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT  
 Cross Validation at 10 fold  
 fitting 10 folds for each of 10 candidates, totalling 100 fits

```
RandomizedSearchCV(cv=10,
                  estimator=RandomForestClassifier(), n_jobs=4,
                  param_distributions={'bootstrap': [True, False],
                                      'criterion': ['gini',
                                                    'entropy'],
                                      'max_depth': [2, 4],
                                      'max_features': ['auto',
                                                       'sqrt'],
                                      'min_samples_leaf': [1, 2],
                                      'min_samples_split': [2,
                                                            5],
```



```
'n_estimators': [10, 17, 25,
33, 41, 48,
56, 64, 72,
80]},
verbose=2)
```

### Best Parameter Result Generated From the Parameter Range Provided

```
rf_RandomGrid.best_params_
{'n_estimators': 72,
'min_samples_split': 5,
'min_samples_leaf': 2,
'max_features': 'sqrt',
'max_depth': 4,
'criterion': 'entropy',
'bootstrap': True}
```

Optimized Hyperparameter Tuning Of Random Forest Classifier Result  
Train Accuracy - : 97.833%

### COMPARATIVE ANALYSIS RESULT WITH OTHER RELATED MACHINE LEARNING ALGORITHM

The optimized value (accuracy) obtained from this research work is later compared with other algorithm. The results is shown below

Naive Bayes Algorithm Result

```
fromsklearn.model_selectionimporttrain_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,
est_size=0.2,random_state=9)#Split the dataset
fromsklearn.naive_bayesimportGaussianNB
nv=GaussianNB()# create a classifier
nv.fit(X_train,y_train)# fitting the data
fromsklearn.metricsimportaccuracy_score
y_pred=nv.predict(X_test)# store the prediction
data
#accuracy_score(y_test,y_pred)# calculate the
accuracy
print("Accuracy of Naive Bayes Algorithm is :
{}".format(accuracy_score(y_test,y_pred)*100))
Accuracy of Naive Bayes Algorithm is :
52.92716808890653
```

Logistic Regression

```
importmatplotlib.pyplotasplt
importnumpyas np
fromsklearn.linear_modelimportLogisticRegression
```

```
fromsklearn.metricsimportclassification_report,
confusion_matrix
model=LogisticRegression(solver='liblinear',
random_state=0)
model.fit(X, y)
LogisticRegression(C=1.0, class_weight=None,
dual=False, fit_intercept=True,
intercept_scaling=1, l1_ratio=None, max_iter=100,
multi_class='warn', n_jobs=None, penalty='l2',
random_state=0, solver='liblinear', tol=0.0001,
verbose=0,
warm_start=False)
model=LogisticRegression(solver='liblinear',
random_state=0).fit(X, y)
model.predict(X)
model.score(X, y)*100
Accuracy: 88.57215435053544
```

RANDOM FOREST CLASSIFIER

```
fromsklearn.ensembleimportRandomForestClassifier
```

```
fromsklearn.metricsimportconfusion_matrix
fromsklearn.metricsimportclassification_report
fromsklearn.metricsimportaccuracy_score
rf=RandomForestClassifier(n_estimators=50,min_s
amples_leaf=0.2,random_state=42)
rf.fit(X_train,y_train)
pred=rf.predict(X_test)
print("Accuracy of Random Forest model is :
{}".format(accuracy_score(y_test,pred)*100))
Accuracy of Random Forest model is : 91.82
```

SUPPORT VECTOR MACHINE

```
fromsklearn.model_selectionimporttrain_test_split
X_train, X_test, y_train, y_test=train_test_split(X,
y, test_size=0.2)
fromsklearn.svmimport SVC
svclassifier= SVC(kernel='rbf', degree=8)
svclassifier.fit(X_train, y_train)
y_pred=svclassifier.predict(X_test)
fromsklearn.metricsimportclassification_report,
confusion_matrix
fromsklearn.metricsimportaccuracy_score
print("Accuracy of the Support Vector Machine
model is : {}".format(accuracy_score(y_test,y_pred
)*100))
Accuracy of the Support Vector Machine model is :
53.70
```

S/N	ESTIMATORS	MIN. SAMPLE LEAF	RANDOM STATE	ACCURACY (%)
1	100	80	50	98.80
2	200	150	100	98.55
3	200	250	150	98.32
4	500	250	200	98.32
5	500	300	250	97.67
6	50	30	32	99.36
7	50	45	42	99.24
8	50	25	32	99.37
9	30	18	21	99.48
10	20	13	25	99.51
11	10	8	15	99.57
12	5	3	6	99.63
13	5	2	3	99.68

Table 1. 1. Experimental summary on manual hyperparameter tuning.

## II. DISCUSSION ON TABLE 1

This section of experiment was carried out with selected numbers of hyperparameter values which included the numbers of estimators, minimum sample leaf and random state, results of the experiments clearly stated below the table. When the value of the estimator was set at 500, Min. Sample leaf 300 and Random state 250, it was observed the performance in outcome with respect to accuracy declined, which implies that the

greater the numbers of trees on the nodes, the less predictive the accuracy of the outcome. It was also observed that when the estimator value was set on 5, Min. Sample leaf 2 and Random state at 3, the efficiency on the outcome was greatly achieved.

This is just a phase testing of my model to check for correctness on predictive purpose, and this is also a manual phase on my software module.

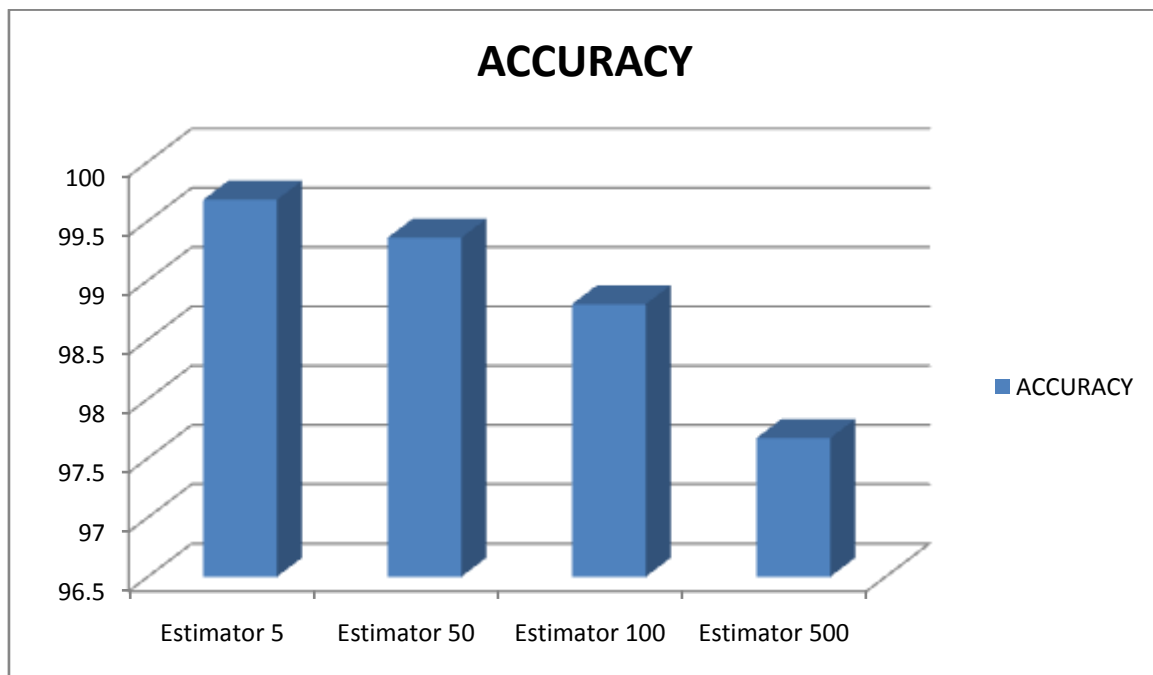


Fig 4.3 Graph depicting accuracy level of the manual hyperparameter tuning.

ESTIMATORS	ACCURACY
Estimator 5	99.68
Estimator 50	99.36
Estimator 100	98.8
Estimator 500	97.67

### REFERENCES

- [1]. Barreiros M, Lundqvist P (2011). QoS-Enabled Networks: Tools and Foundations. West Sussex, UK: John Wiley & Sons.
- [2]. Bhoyar, R., Ghonge, M., Gupta, S., 2013. Comparative study on IEEE standard of wireless LAN/Wi-Fi 802.11 a/b/g/n. Int. J. Adv. Res. Electron. Commun. Eng. (IJARECE) 2 (7).
- [3]. Chiu DM, Sudama R (1992). Network monitoring explained: design and application. Ellis Horwood Series in Computer Communications and Networking.
- [4]. Cisco (2008). Performance Management Best Practices for Broadband Service Providers. USA: Cisco Systems Inc.
- [5]. Feng, P. 2012. Wireless LAN security issues and solutions. In: Robotics and Applications (ISRA), 2012 IEEE Symposium on (pp. 921–924). IEEE.
- [6]. Gokhale AA (2005). Introduction to Telecommunications (2 ed.). New York, United States of America: Thomson Delmer Learning.
- [7]. Haykin S (2001). Communication Systems (4 ed.). New York, USA: John Wiley & Sons, Inc.
- [8]. Hong J, Li VOK (2009). Impact of Information on Network Performance – An Information- Theoretic Perspective. IEEE Glob. Telecomm.Conf. Publ. pp.1-6.
- [9]. Keiser G (2002). Local Area Networks. New York, United States of America: McGraw-Hill Companies.
- [10]. Koendjibiarie S, Koppius O, Vervest P, van Heck E (2010). Network transparency and the performance of dynamic business networks.2010 4th IEEE International Conference on Digital Ecosystems and Technologies (DEST).pp.197-202.
- [11]. Kouvastos DD (2011). Network Performance Engineering: A Handbook on Convergent Multi-service Networks and Next Generation Internet. German: Springer-Verlag Berlin Heidelberg.
- [12]. Lathi, B. (1998). Modern Digital and Analog Communication Systems. New York: Oxford University Press Inc.
- [13]. Lawniczak AT, Tang X (2006). Packet Switching Network Performance Indicators as Function of Network Topology and Routing Algorithms.IEEE Conference (CCECE '06) Publication.pp.1008-1011.
- [14]. Milliken WC (2005). Patent No. 6978223. USA.
- [15]. Nassar DJ (2000). Network Performance Baselining (1 ed.). USA: MTP.
- [16]. Park KI (2005). QoS in Packet Networks. USA: Springer.
- [17]. Prasad, RS, Murray M, Dovrolis C, Claffy K (2003). Bandwidth estimation: metrics, measurement and tools. Networking Journal, IEEE.
- [18]. Seshan S, Stemm M, Katz RH (1997). SPAND: Shared Passive Network Performance Discovery. USENIX Symposium on Internet Technologies and Systems. California: USENIX.
- [19]. Sheldon, F.T., Weber, J.M., Yoo, S.M., Pan, W.D., 2012. Theinsecurity of wireless networks.Secur. Privacy IEEE 10 (4), 54–61
- [20]. Soldani D, Li M, Cuny R (2006). QoS and QoE Management in UMTS Cellular Systems. West Sussex, UK: John Wiley & Sons.
- [21]. Spohn DL (2000). Data Network Design. New York, United States of America: McGraw-Hill Companies Inc.
- [22]. (22) Stallings W (1996July9). Knowing wiring basics can boost local net performance. Network World , P. 29.
- [23]. ) Stanford Linear Accelerator Center (SLAC). (n.d.).Network Monitoring Tools.Retrieved August 29, 2013, from <http://www.slac.stanford.edu/xorg/nmtf/nmtf-tools.html>.
- [24]. Walrand J, Varaiya P (2000). High-Performance Communication Networks. USA: Academic Press.

- [25]. Ogunrinde, S. I., 2004. Network Programming and Design, Heinemann Educational Press, pp. 1-12
- [26]. Menkiti, A. I., 2005. Logic Circuits, Devices and Applications, EFTIMO Nig Press, Calabar, pp 82-90
- [27]. Yucalar, F. & Erdogan, S. Z., July 2009. A Questionnaire Based Method for CMMI Level 2 Maturity Assessment. Journal of aeronautics and Space Technologies, pp. 39-46
- [28]. Vincent A. Akpan, Reginald O. A. Osakwe and AmakuAmaku, Efficient Networking Of Tini For Real-Time Weather Data Logging& Deployment Over Ethernet And Serialcommunication Links. International Journal of Communications, Network and System Sciences (IJCNS), September 2013, P. O. BOX 54821, Irvine CA 92619-4821, USA. (PUBLISHED)
- [29]. AmakuAmaku, Raphael E.Watti, John Joshua, Optic Fiber As A Reliable Medium For Metropolitan Area Networking (Man) Connectivity, International Journal of Engineering and Technology (IJET) Volume 4 No. 7, July, 2014, ISSN: 2049-3444 © 2014 –IJET Publications UK.
- [30]. AmakuAmaku, RaphealWatti, Igbinsosa G., Bandwidth As A Determinant Factor For Effective Internet Connectivity In Educational Research Purpose In Higher Institution Of Learning. International Journal of Engineering and Technology (IJET) Volume 6 No. 4, April , 2016, ISSN: 2049-3444 © 2016 –IJET Publications UK.
- [31]. M.I. Jordan, T.M. Mitchell, Machine learning: Trends, perspectives, and prospects, Science 349 (2015) 255260. <https://doi.org/10.1126/science.aaa8415>.
- [32]. M.-A. Ziller and M. F. Huber, Benchmark and Survey of Automated Machine Learning Frameworks, arXiv preprint arXiv:1904.12054, (2019). <https://arxiv.org/abs/1904.12054>.
- [33]. R. E. Shawi, M. Maher, S. Sakr, Automated machine learning: State-of-the-art and open challenges, arXiv preprint arXiv:1906.02287, (2019). <http://arxiv.org/abs/1906.02287>.
- [34]. M. Kuhn and K. Johnson, Applied Predictive Modeling, Springer (2013) ISBN: 9781461468493.
- [35]. G.I. Diaz, A. Fokoue-Nkoutche, G. Nannicini, H. Samulowitz, An effective algorithm for hyperparameter optimization of neural networks, IBM J. Res. Dev. 61 (2017) 120. <https://doi.org/10.1147/JRD.2017.2709578>.
- [36]. F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., Automatic Machine Learning: Methods, Systems, Challenges, Springer (2019) ISBN 9783030053185.
- [37]. N. Decastro-Garca, . L. MuozCastaeda, D. EscuderoGarca, and M. V. Carriegos, Effect of the Sampling of a Dataset in the Hyperparameter Optimization Phase over the Efficiency of a Machine Learning Algorithm, Complexity 2019 (2019). <https://doi.org/10.1155/2019/6278908>.
- [38]. S. Abreu, Automated Architecture Design for Deep Neural Networks, arXiv preprint arXiv:1908.10714, (2019). <http://arxiv.org/abs/1908.10714>.
- [39]. O. S. Steinholtz, A Comparative Study of Black-box Optimization Algorithms for Tuning of Hyper-parameters in Deep Neural Networks, M.S. thesis, Dept. Elect. Eng., Lule Univ. Technol., (2018).
- [40]. G. Luo, A review of automatic selection methods for machine learning algorithms and hyper-parameter values, Netw. Model.Anal. Heal, Informatics Bioinforma. 5 (2016) 116. <https://doi.org/10.1007/s13721-016-0125-6>.
- [41]. D. Maclaurin, D. Duvenaud, R.P. Adams, Gradient-based Hyper-parameter Optimization through Reversible Learning, arXiv preprint arXiv:1502.03492, (2015). <http://arxiv.org/abs/1502.03492>.
- [42]. J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, Algorithms for hyper-parameter optimization, Proc. Adv. Neural Inf. Process. Syst., (2019)25462554.
- [43]. B. James and B. Yoshua, Random Search for Hyper-Parameter Optimization, J. Mach. Learn. Res. 13 (1) (2019) 281305.
- [44]. N. Landwehr, M. Hall, and E. Frank, —Logistic model trees, Mach. Learn., vol. 59, no. 12, pp. 161-205, 2015.
- [45]. Breiman Leo (2001). "Random Forests". Machine Learning 45 (1): 5–32.
- [46]. Liaw, Andy (16 October 2012). "Documentation for R package random forest". Retrieved 15 March 2013.