



# OPTIMIZATION OF A HYBRID-BASED RANDOM FOREST ALGORITHM FOR NETWORK SYSTEMS USING RANDOMIZED SEARCH HYPERPARAMETER TUNING METHOD.

Amaku Amaku<sup>1</sup>, Olumide Owolabi<sup>2</sup>, Agbogun B. Joshua<sup>3</sup>, Bamidele, Oluchi Jennie<sup>4</sup>, Igbinsosa O. G<sup>5</sup>  
1Department of Computer Science, Godfrey Okoyo University, Enugu State, Nigeria.

2Department of Computer Sciences, University of Abuja, Abuja, Nigeria.

3Department of Mathematics and Computer Science, Godfrey Okoyo University, Enugu State, Nigeria.

4Computer Science Programme, National Mathematical Centre, Kwali, Abuja

5Department of Computer Science, College of ICT, Salem University, P.M.B. 1060 Lokoja, Kogi State, Nigeria.

## ABSTRACT

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 are 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. This work further remodels the Random Forest algorithm using RandomizedSearchCV method hyperparameter tuning as base-behavioral classifier to check and compare with its default in terms of efficiency in the realm of machine learning. NSL-KDD dataset were used for both training and testing of the tuned model using a supervised learning method. The predictive performance in the tuned model with respect to its matrix was higher, and comparison with other algorithms like Naïve bayes and Perception model, Ridge classifier

proved that the RandomizedSearchCV hyperparameter tuning Random Forest algorithm performed more efficiently its results analysis and computation.

Keywords: RandomizedSearchCV, Hyperparameter, Decision Tree, Classifier, Random forest, Optimization, Tuning.

## DECISION TREE

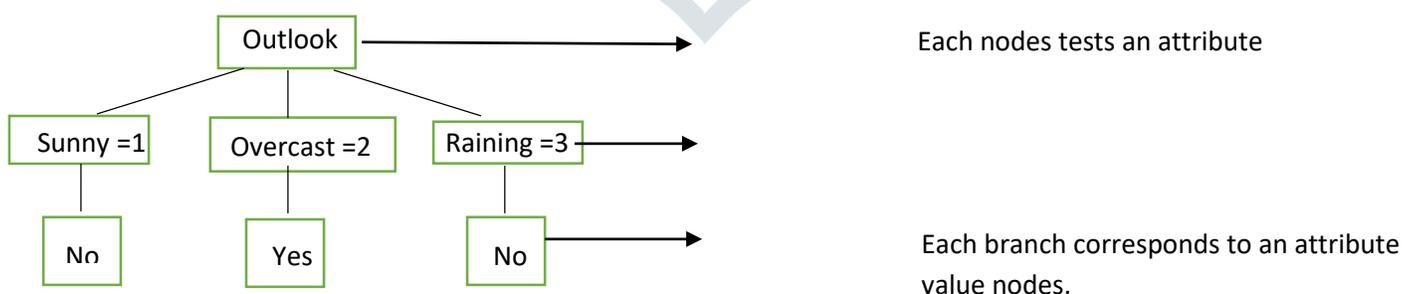
Decision Tree is a graphical representation of all possible solutions to a decision, decision tree is based on some conditions and it can be easily explained. It represents a function that takes as Input a vector of attribute values and returns a “decision” – a single output value.

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

### For Example



Figure

### 1.1 Decision Tree Learning Algorithm

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

- ID<sub>3</sub> is one of the most common decision tree algorithms.
- 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 calculates 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:

$$\text{Entropy}(s) = - \sum P(I) \cdot \log_2 P(I) \dots\dots\dots (1)$$

$$\text{Gain}(S,A) = \text{Entropy}(s) - \sum [P(S/A) \cdot \text{Entropy}(S/A)] \dots\dots\dots (2)$$

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 feature's 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 trees, 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).

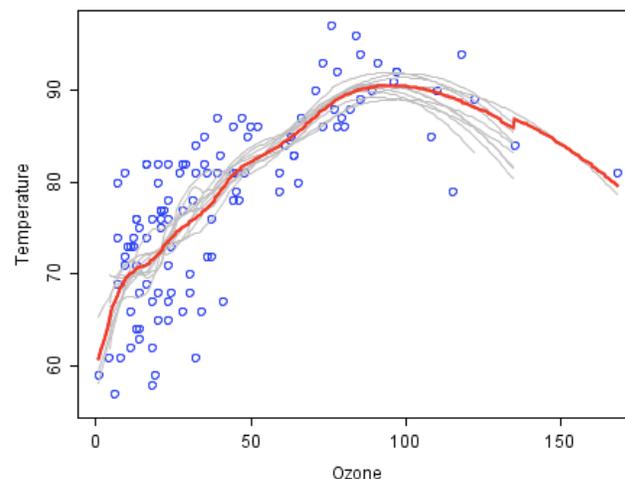
Given the algorithm for choosing decision stumps, the algorithm for growing larger decision trees is straightforward. We begin by selecting the overall best decision stump for the classification task. We then check the accuracy of each of the leaves on the training set. Leaves that do not achieve sufficient 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.

## RANDOM FOREST

The random forest (Breiman, 2001) is an ensemble approach that can also be thought of as a form of nearest neighbor predictor.

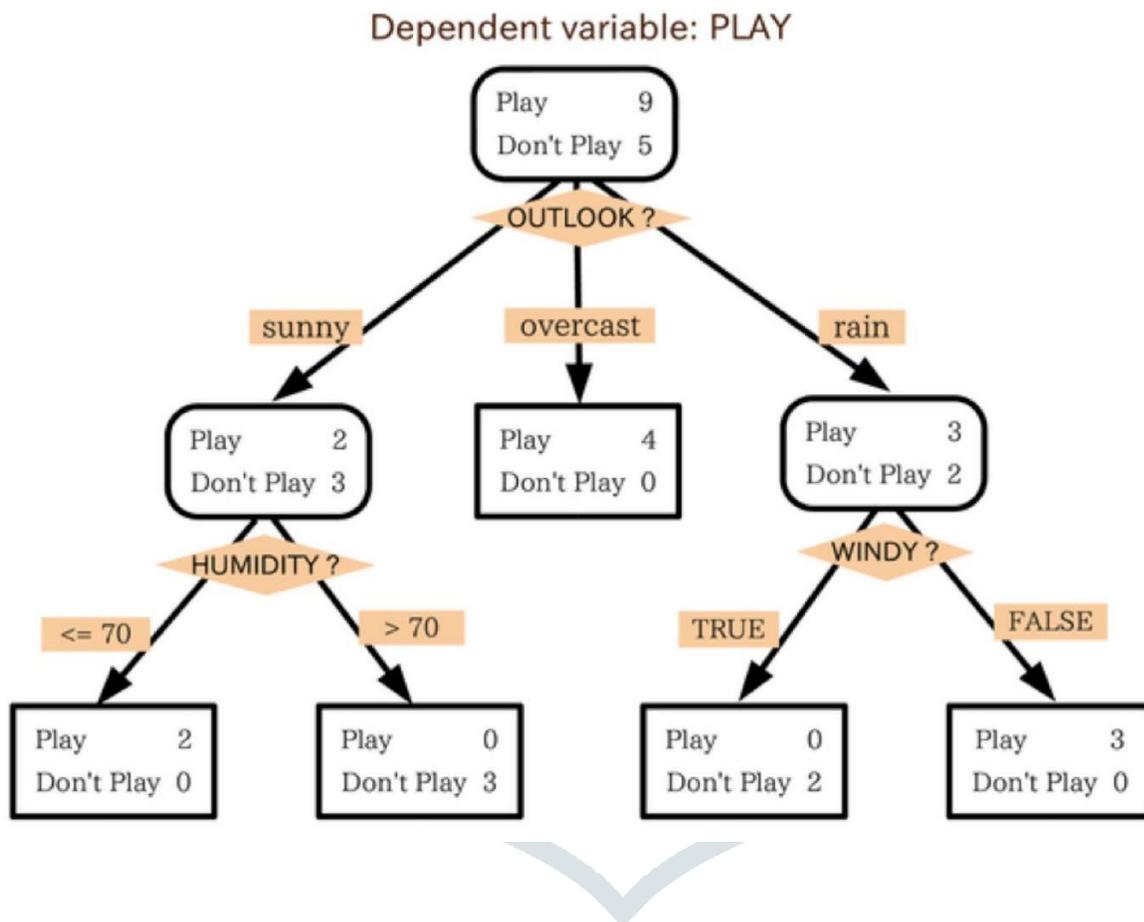
Ensembles are a divide-and-conquer approach used to improve performance. The main principle behind ensemble methods is that a group of "weak learners" can come together to form a "strong learner". The figure below provides an example. Each classifier, individually, is a "weak learner," while all the classifiers taken together are a "strong learner".

The data to be modeled are the blue circles. One can assume that they represent some underlying function plus noise. Each individual learner is shown as a gray curve. Each gray curve (a weak learner) is a fair approximation to the underlying data. The red curve (the ensemble "strong learner") can be seen to be a much better approximation to the underlying data.



**Trees and Forests:** The random forest starts with a standard machine learning technique called a “decision tree” which, in ensemble terms, corresponds to the to weak learner. In a decision tree, an input is entered at the top and as it traverses down the tree the data gets bucketed into smaller and smaller sets. For details see the figure below is taken.

In this example, the tree indicates that, based upon weather conditions, whether to play ball. For example, if the outlook is sunny and the humidity is less than or equal to 70, then it’s probably OK to play.



The random forest takes this notion to the next level by combining trees with the notion of an ensemble. Thus, in ensemble terms, the trees are weak learners and the random forest is a strong learner.

Here is how such a system is trained; for some number of trees  $T$ :

1. Sample  $N$  cases at random with replacement to create a subset of the data. The subset should be about 66% of the total set.
2. At each node:
  1. For some number  $m$ ,  $m$  predictor variables need to be selected at random from all the predictor variables.

2. The predictor variable that provides the best split, according to some objective function, is used to do a binary split on that node.
3. At the next node, choose another  $m$  variables at random from all predictor variables and do the same.

Depending upon the value of  $m$ , there are three slightly different systems:

- Random splitter selection:  $m = 1$
- Breiman's bagger:  $m =$  total number of predictor variables
- Random forest:  $m \ll$  number of predictor variables. Brieman suggests three possible values form:  $\frac{1}{2}\sqrt{m}$ ,  $\sqrt{m}$ , and  $2\sqrt{m}$

**Running a Random Forest.** When a new input is entered into the system, it is run down all of the trees. The result may either be an average or weighted average of all of the terminal nodes that are reached, or, in the case of categorical variables, a voting majority.

Note that:

- With a large number of predictors, the eligible predictor set will be quite different from node to node.
- The greater the inter-tree correlation, the greater the random forest error rate, so one pressure on the model is to have the trees as uncorrelated as possible.
- As  $m$  goes down, both inter-tree correlation and the strength of individual trees go down. So some optimal value of  $m$  must be discovered.

**Strengths and weaknesses:** Random forest runtimes are quite fast, and they are able to deal with unbalanced and missing data. A Random Forest weakness is that when used for regression they cannot predict beyond the range in the training data, and that they may over-fit data sets that are particularly noisy. Of course, the best test of any algorithm is how well it works upon your own data set.

## GENERAL OVERVIEW RANDOM FOREST

Random Forests Random forests is a idea of the general technique of random decision forests that are an ensemble learning technique for classification, regression and other tasks, that control by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests accurate for decision trees' habit of over fitting to their training set and the first algorithm for random decision forests was created by Tin Kam Ho using the random subspace method which, in Ho's formulation, is a way to implement the "stochastic discrimination"

approach to classification proposed by Eugene Kleinberg. An extension of the algorithm was developed by Leo Breiman and Adele Cutler, and "Random Forests" is their trademark (Landwehr et al,2015). The extension combines Breiman's "bagging" idea and random selection of features, introduced first by Ho and later independently by Amit and Geman in order to construct a collection of decision trees with controlled variance (Breiman, 2001). Random Tree is a supervised Classifier; it is an ensemble learning algorithm that generates lots of individual learners. It employs a bagging idea to construct a random set of data for constructing a decision tree. In standard tree every node is split using the best split among all variables. In a random forest, every node is split using the best among the subset of predictors randomly chosen at that node. Random trees have been introduced by Leo Breiman and Adele Cutler (Liaw, 2013). The algorithm can deal with both classification and regression problems.

## RANDOM TREES

A random tree is a group (ensemble) of tree predictors that is called forest. The classification mechanisms as follows: the random trees classifier gets the input feature vector, classifies it with every tree in the forest, and outputs the class label that received the majority of "votes". In case of a regression, the classifier reply is the average of the responses over all the trees in the forest. Random Trees are essentially the combination of two existing algorithms in Machine Learning: single model trees are merged with Random Forest ideas (Liaw, 2013). Model trees are decision trees where every single leaf holds a linear model which is optimized for the local subspace explained by this leaf. Random Forests have shown to improve the performance of single decision trees considerably (Landwehr, 2015).

First the training data is sampled with replacement for each single tree like in Bagging and secondly, when growing a tree, instead of always computing the best possible split for each node only a random subset of all attributes is considered at every node, and the best split for that subset is computed. Such trees have been for classification Random model trees for the first time combine model trees and random forests. Random trees uses this produce for split selection and thus induce reasonably balanced trees where one global setting for the ridge value works across all leaves, thus simplifying the optimization procedure (Liaw, 2013).

### 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  $P(\text{sunny})$  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  $\log_2 P(l)$  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  $\log_2 P(l)$  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 are 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 reliably 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 fact 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.

### 3 Model Design Phase

Hyperparameters are different from the internal model parameters, such as the neural network's weights, which can be learned from the data during the model training phase. Before the training phase, a set of hyperparameter values is entered which archive the best performance on the data in a reasonable amount of time. This process is called hyperparameter optimization or tuning. It plays a vital role in the prediction accuracy of machine learning algorithms. There are mainly two kinds of hyperparameter optimization methods, i.e., manual search and automatic search methods. Manual search tries out hyperparameter sets by hand. It depends on the fundamental intuition and experience of expert users who can identify the important parameters that have a greater impact on the results and then determine the relationship between certain parameters and final results through the

visualization tools (Aarshay, 2018). Manual search requires users to have more professional background knowledge and practical experience. And it is hard to be applied by non-expert users. The process of tuning hyperparameters is not easily reproducible. Besides, as the number of hyperparameters and the range of values increase, it becomes quite difficult to manage since humans are not good at handling high dimensional data and easily misinterpret or miss trends and relationships in hyperparameters. To overcome the drawbacks of manual search, automatic search algorithms have been proposed, such as grid search, randomized search (Bergstra, 2012) or Cartesian hyperparameter search. The principle of grid search is exhaustive searching. Grid search trains a machine learning model with each combination of possible values of hyperparameters on the training set and evaluates the performance according to a predefined metric on a cross validation set. Although this method achieves automatic tuning and can theoretically obtain the global optimal value of the optimization objective function, it suffers from the curse of dimensionality, i.e., the efficiency of the algorithm decreases rapidly as the number of hyperparameters being tuned and the range of values of hyperparameters increase. To solve the problem of expensive cost in grid search, the random search algorithm (Bergstra, 2012) has been proposed, which found that for most data sets, only a few of the hyperparameters really matter. The overall efficiency can be improved by reducing the search to hyperparameters that do not matter, and finally the approximate solution of the optimization function is obtained.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	'duration'	'protocol'	'service'	'flag'	'src_bytes'	'dst_bytes'	'land'	'wrong_fr'	'urgent'	'hot'	'num_fail'	'logged_ir'	'num_con'	'root_shel'	'su_attempt'	'num_roo'	'num_file'	'num_she'	'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	8153	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	REJ	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_jc	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	REJ	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	616	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	mtp	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	5607	udp	other	SF	147	105	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

## 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)x
```

## 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,
```

```
                                         56, 64, 72, 80]}),
```

48,

```
    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

### M

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

Naive Bayes Algorithm Result

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=9) #Split the dataset
from sklearn.naive_bayes import GaussianNB
nv = GaussianNB() # create a classifier
nv.fit(X_train,y_train) # fitting the data
from sklearn.metrics import accuracy_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

```
import matplotlib.pyplot as plt
```

```
import numpy as np

from sklearn.linear_model import LogisticRegression

from sklearn.metrics import classification_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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

from sklearn.metrics import accuracy_score

rf=RandomForestClassifier(n_estimators=50,min_samples_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

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
```

```
from sklearn.svm import SVC
```

```
svclassifier = SVC(kernel='rbf', degree=8)
```

```
svclassifier.fit(X_train, y_train)
```

```
y_pred = svclassifier.predict(X_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix
```

```
from sklearn.metrics import accuracy_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

## K-Nearest Neighbor Algorithm

```
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import accuracy_score
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 100)
```

```
KNeighborsClassifier(
```

```
    n_neighbors=5,      # The number of neighbours to consider
```

```
    weights='uniform', # How to weight distances
```

```
    algorithm='auto',  # Algorithm to compute the neighbours
```

```
    leaf_size=30,      # The leaf size to speed up searches
```

```
    p=2,               # The power parameter for the Minkowski metric
```

```
    metric='minkowski', # The type of distance to use
```

```
    metric_params=None, # Keyword arguments for the metric function
```

```
    n_jobs=None        # How many parallel jobs to run
```

```
)  
clf = KNeighborsClassifier(p=1)  
clf.fit(X_train, y_train)  
predictions = clf.predict(X_test)  
print(accuracy_score(y_test, predictions))  
0.89
```

## AMAKU'S MODEL ANALYSIS

Research title: OPTIMIZATION OF BEHAVIORAL BASED RANDOM FOREST ALGORITHM AS A MACHINE LEARNING TOOL IN INTRUSION DETECTION SYSTEM

Importing necessary libraries

```
import pandas as pd  
import numpy as np  
import sys
```

## DATASET IMPORTATION USING PANDAS

```
#from google.colab import files  
#uploaded = files.upload()  
  
#import io  
#df = pd.read_csv(io.BytesIO(uploaded['KDDTrain+.csv']))
```

```
df=pd.read_csv('KDDTrain+.csv')
```

```
df.head()
```

## STATISTICAL SUMMARY

```
df.describe()
```

## DATA PREPROCESSING

```
# adding column labels
```

```
columns = ([ 'duration'  
, 'protocol_type'  
, 'service'  
, 'flag'  
, 'src_bytes'  
, 'dst_bytes'  
, 'land'  
, 'wrong_fragment'  
, 'urgent'  
, 'hot'  
, 'num_failed_logins'  
, 'logged_in'  
, 'num_compromised'  
, 'root_shell'  
, 'su_attempted'  
, 'num_root'  
, 'num_file_creations'  
, 'num_shells'  
, 'num_access_files'  
, 'num_outbound_cmds'  
, 'is_host_login'  
, 'is_guest_login'  
, 'count'  
, 'srv_count'  
, 'serror_rate'  
, 'srv_serror_rate'
```



```
, 'error_rate'  
, 'srv_error_rate'  
, 'same_srv_rate'  
, 'diff_srv_rate'  
, 'srv_diff_host_rate'  
, 'dst_host_count'  
, 'dst_host_srv_count'  
, 'dst_host_same_srv_rate'  
, 'dst_host_diff_srv_rate'  
, 'dst_host_same_src_port_rate'  
, 'dst_host_asrv_diff_host_rate'  
, 'dst_host_serror_rate'  
, 'dst_host_srv_serror_rate'  
, 'dst_host_rerror_rate'  
, 'dst_host_srv_rerror_rate'  
, 'class']])
```

```
df.columns = columns
```

```
#test_df.columns = columns
```

```
# sanity check
```

```
df.head()
```

```
X = df.drop(columns=['protocol_type', 'service', 'flag', 'class'])
```

```
y = df['class'] # the last column in the dataset is used as y value
```

```
y.head()
```



```
X.head()
```

Cross Validation: Accuracy, Precision, Recall, F-measure RANDOM FOREST CLASSIFIER

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6, random_state=42)
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import classification_report
```

```
from sklearn.metrics import accuracy_score
```

```
rf=RandomForestClassifier(n_estimators=50,min_samples_leaf=0.2,random_state=42)
```

```
rf.fit(X_train,y_train)
```

```
pred=rf.predict(X_test)
```

```
print ('CONFUSION MATRIX')
```

```
print(confusion_matrix(y_test, pred))
```

```
print ("")
```

```
print ('CLASSIFICATION REPORT')
```

```
print(classification_report(y_test,pred))
```

```
print("Accuracy of the given model is : {}".format(accuracy_score(y_test,pred)*100))
```

```
#'random_forest': {
```

```
# 'model': RandomForestClassifier(),
```

```
# 'params' : {
```

```
# 'n_estimators': [1,5,10]
```

```
# }
```

```
#}
```

```
# 'params': {
```

```
# 'C': [1,5,10]
#}
#}
#}
#}
#scores = []

#for model_name, mp in model_params.items():
# clf = GridSearchCV(mp['model'], mp['params'], cv=5, return_train_score=False)
# clf.fit(X, y)
# scores.append({
# 'model': model_name,
# 'best_score': clf.best_score_,
# 'best_params': clf.best_params_
#})

#df = pd.DataFrame(scores,columns=['model','best_score','best_params'])

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

from sklearn.metrics import accuracy_score

rf=RandomForestClassifier(n_estimators=5,min_samples_leaf=2,random_state=3)

rf.fit(X_train,y_train)

pred=rf.predict(X_test)

print ('CONFUSION MATRIX')

print(confusion_matrix(y_test, pred))
```

```
print("")
print('CLASSIFICATION REPORT')
print(classification_report(y_test,pred))
print("Accuracy of the given model is : {}".format(accuracy_score(y_test,pred)*100))
```

CONFUSION MATRIX

```
[[38541  166]
 [ 99 44337]]
```

CLASSIFICATION REPORT

	precision	recall	f1-score	support
anomaly	1.00	1.00	1.00	38707
normal	1.00	1.00	1.00	44436
accuracy			1.00	83143
macro avg	1.00	1.00	1.00	83143
weighted avg	1.00	1.00	1.00	83143

Accuracy of the given model is : 99.6812720253058

Fig: 4.2..1 Experiment 1 on selected hyperparameter tuning

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import classification_report
```

```
from sklearn.metrics import accuracy_score
```

```
rf=RandomForestClassifier(n_estimators=100,min_samples_leaf=80,random_state=50)
```

```

rf.fit(X_train,y_train)

pred=rf.predict(X_test)

print ('CONFUSION MATRIX')

print(confusion_matrix(y_test, pred))

print ("")

print ('CLASSIFICATION REPORT')

print(classification_report(y_test,pred))

print("Accuracy of the given model is : {}".format(accuracy_score(y_test,pred)*100))

```

CONFUSION MATRIX

```

[[37851  856]
 [ 139 44297]]

```

CLASSIFICATION REPORT

	precision	recall	f1-score	support
anomaly	1.00	0.98	0.99	38707
normal	0.98	1.00	0.99	44436
accuracy		0.99	0.99	83143
macro avg	0.99	0.99	0.99	83143
weighted avg	0.99	0.99	0.99	83143

Accuracy of the given model is : 98.80326666105384

Fig: 4.2..2 Experiment 2 on selected hyperparameter tuning

```

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

from sklearn.metrics import accuracy_score

rf=RandomForestClassifier(n_estimators=200,min_samples_leaf=150,random_state=100)

rf.fit(X_train,y_train)

pred=rf.predict(X_test)

print ('CONFUSION MATRIX')

print(confusion_matrix(y_test, pred))

print ("")

print ('CLASSIFICATION REPORT')

print(classification_report(y_test,pred))

print("Accuracy of the given model is : {}".format(accuracy_score(y_test,pred)*100))

```

CONFUSION MATRIX

[[37644 1063]

[ 141 44295]]

CLASSIFICATION REPORT

precision recall f1-score support

anomaly 1.00 0.97 0.98 38707

normal 0.98 1.00 0.99 44436

accuracy 0.99 83143

macro avg 0.99 0.98 0.99 83143

weighted avg 0.99 0.99 0.99 83143



Accuracy of the given model is : 98.55189252252143

Fig: 4.2.3 Experiment 3 on selected hyperparameter tuning

```

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

from sklearn.metrics import accuracy_score

rf=RandomForestClassifier(n_estimators=200,min_samples_leaf=250,random_state=150)

rf.fit(X_train,y_train)

pred=rf.predict(X_test)

print ('CONFUSION MATRIX')

print(confusion_matrix(y_test, pred))

print ("")

print ('CLASSIFICATION REPORT')

print(classification_report(y_test,pred))

print("Accuracy of the given model is : {}".format(accuracy_score(y_test,pred)*100))

```

CONFUSION MATRIX

[[37492 1215]

[ 177 44259]]

CLASSIFICATION REPORT

precision recall f1-score support

anomaly 1.00 0.97 0.98 38707

normal	0.97	1.00	0.98	44436
accuracy		0.98	83143	
macro avg	0.98	0.98	0.98	83143
weighted avg	0.98	0.98	0.98	83143

Accuracy of the given model is : 98.32577607254971

Fig: 4.2.4 Experiment 4 on selected hyperparameter tuning

```

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

rf=RandomForestClassifier(n_estimators=500,min_samples_leaf=250,random_state=200)

rf.fit(X_train,y_train)

pred=rf.predict(X_test)

print ('CONFUSION MATRIX')

print(confusion_matrix(y_test, pred))

print ("")

print ('CLASSIFICATION REPORT')

print(classification_report(y_test,pred))

print("Accuracy of the given model is : {}".format(accuracy_score(y_test,pred)*100))

CONFUSION MATRIX

[[37491 1216]

 [ 175 44261]]

```

## CLASSIFICATION REPORT

	precision	recall	f1-score	support
anomaly	1.00	0.97	0.98	38707
normal	0.97	1.00	0.98	44436
accuracy		0.98		83143
macro avg	0.98	0.98	0.98	83143
weighted avg	0.98	0.98	0.98	83143

Accuracy of the given model is : 98.32697881962402

Fig: 4.2.5 Experiment 5 on selected hyperparameter tuning

```

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

from sklearn.metrics import accuracy_score

rf=RandomForestClassifier(n_estimators=500,min_samples_leaf=300,random_state=250)

rf.fit(X_train,y_train)

pred=rf.predict(X_test)

print ('CONFUSION MATRIX')

print(confusion_matrix(y_test, pred))

print ("")

print ('CLASSIFICATION REPORT')

print(classification_report(y_test,pred))

```

```
print("Accuracy of the given model is : {}".format(accuracy_score(y_test,pred)*100))
```

CONFUSION MATRIX

```
[[36958 1749]
```

```
[ 190 44246]]
```

CLASSIFICATION REPORT

	precision	recall	f1-score	support
anomaly	0.99	0.95	0.97	38707
normal	0.96	1.00	0.98	44436
accuracy		0.98		83143
macro avg	0.98	0.98	0.98	83143
weighted avg	0.98	0.98	0.98	83143

Accuracy of the given model is : 97.6678734228979

Fig: 4.2.6 Experiment 6 on selected hyperparameter tuning

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import classification_report
```

```
from sklearn.metrics import accuracy_score
```

```
rf=RandomForestClassifier(n_estimators=50,min_samples_leaf=30,random_state=32)
```

```
rf.fit(X_train,y_train)
```

```
pred=rf.predict(X_test)
```

```

print ('CONFUSION MATRIX')

print(confusion_matrix(y_test, pred))

print ("")

print ('CLASSIFICATION REPORT')

print(classification_report(y_test,pred))

print("Accuracy of the given model is : {}".format(accuracy_score(y_test,pred)*100))

```

CONFUSION MATRIX

```
[[38289  418]
```

```
[ 107 44329]]
```

CLASSIFICATION REPORT

	precision	recall	f1-score	support
anomaly	1.00	0.99	0.99	38707
normal	0.99	1.00	0.99	44436
accuracy			0.99	83143
macro avg	0.99	0.99	0.99	83143
weighted avg	0.99	0.99	0.99	83143

Accuracy of the given model is : 99.36855778598319

Fig: 4.2.7 Experiment 7 on selected hyperparameter tuning

```

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix

```

```

from sklearn.metrics import classification_report

from sklearn.metrics import accuracy_score

rf=RandomForestClassifier(n_estimators=50,min_samples_leaf=45,random_state=42)

rf.fit(X_train,y_train)

pred=rf.predict(X_test)

print ('CONFUSION MATRIX')

print(confusion_matrix(y_test, pred))

print ("")

print ('CLASSIFICATION REPORT')

print(classification_report(y_test,pred))

print("Accuracy of the given model is : {}".format(accuracy_score(y_test,pred)*100))

```

CONFUSION MATRIX

[[38203 504]

[ 125 44311]]

CLASSIFICATION REPORT

	precision	recall	f1-score	support
anomaly	1.00	0.99	0.99	38707
normal	0.99	1.00	0.99	44436
accuracy		0.99		83143
macro avg	0.99	0.99	0.99	83143
weighted avg	0.99	0.99	0.99	83143

Accuracy of the given model is : 99.2434720902541

Fig: 4.2.8 Experiment 8 on selected hyperparameter tuning

```

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

from sklearn.metrics import accuracy_score

rf=RandomForestClassifier(n_estimators=50,min_samples_leaf=45,random_state=42)

rf.fit(X_train,y_train)

pred=rf.predict(X_test)

print ('CONFUSION MATRIX')

print(confusion_matrix(y_test, pred))

print ("")

print ('CLASSIFICATION REPORT')

print(classification_report(y_test,pred))

print("Accuracy of the given model is : {}".format(accuracy_score(y_test,pred)*100))

```

CONFUSION MATRIX

```
[[38298  409]
```

```
[ 108 44328]]
```

CLASSIFICATION REPORT

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

anomaly	1.00	0.99	0.99	38707
---------	------	------	------	-------

normal	0.99	1.00	0.99	44436
--------	------	------	------	-------

accuracy		0.99		83143
----------	--	------	--	-------

macro avg	0.99	0.99	0.99	83143
-----------	------	------	------	-------

weighted avg	0.99	0.99	0.99	83143
--------------	------	------	------	-------



Accuracy of the given model is : 99.37817976257773

Fig: 4.2.9 Experiment 9 on selected hyperparameter tuning

```

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

from sklearn.metrics import accuracy_score

rf=RandomForestClassifier(n_estimators=30,min_samples_leaf=18,random_state=21)

rf.fit(X_train,y_train)

pred=rf.predict(X_test)

print ('CONFUSION MATRIX')

print(confusion_matrix(y_test, pred))

print ("")

print ('CLASSIFICATION REPORT')

print(classification_report(y_test,pred))

print("Accuracy of the given model is : {}".format(accuracy_score(y_test,pred)*100))

```

CONFUSION MATRIX

[[38382 325]

[ 109 44327]]

CLASSIFICATION REPORT

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

anomaly	1.00	0.99	0.99	38707
---------	------	------	------	-------

normal	0.99	1.00	1.00	44436
--------	------	------	------	-------

```

accuracy          0.99  83143
macro avg    0.99  0.99  0.99  83143
weighted avg  0.99  0.99  0.99  83143

```

Accuracy of the given model is : 99.4780077697461

Fig: 4.2.10 Experiment 10 on selected hyperparameter tuning



```

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

rf=RandomForestClassifier(n_estimators=20,min_samples_leaf=13,random_state=25)
rf.fit(X_train,y_train)
pred=rf.predict(X_test)

print ('CONFUSION MATRIX')

print(confusion_matrix(y_test, pred))

print ("")

print ('CLASSIFICATION REPORT')

print(classification_report(y_test,pred))

print("Accuracy of the given model is : {}".format(accuracy_score(y_test,pred)*100))

CONFUSION MATRIX

[[38384  323]
 [  88 44348]]

```

CLASSIFICATION REPORT

	precision	recall	f1-score	support
anomaly	1.00	0.99	0.99	38707
normal	0.99	1.00	1.00	44436
accuracy		1.00		83143
macro avg	1.00	0.99	1.00	83143
weighted avg	1.00	1.00	1.00	83143

Accuracy of the given model is : 99.5056709524554

Fig: 4.2.11 Experiment 11 on selected hyperparameter tuning

```

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion_matrix

from sklearn.metrics import classification_report

from sklearn.metrics import accuracy_score

rf=RandomForestClassifier(n_estimators=10,min_samples_leaf=8,random_state=15)

rf.fit(X_train,y_train)

pred=rf.predict(X_test)

print ('CONFUSION MATRIX')

print(confusion_matrix(y_test, pred))

print ("")

print ('CLASSIFICATION REPORT')

print(classification_report(y_test,pred))

print("Accuracy of the given model is : {}".format(accuracy_score(y_test,pred)*100))

CONFUSION MATRIX

```

[[38446 261]

[ 94 44342]]

## CLASSIFICATION REPORT

	precision	recall	f1-score	support
anomaly	1.00	0.99	1.00	38707
normal	0.99	1.00	1.00	44436
accuracy		1.00	83143	
macro avg	1.00	1.00	1.00	83143
weighted avg	1.00	1.00	1.00	83143

Accuracy of the given model is : 99.5730247886172

Fig: 4.2.12 Experiment 12 on selected hyperparameter tuning

INPUT/TUNE VALUES OF INTEREST HERE(PARAMETER GRID)

*# Number of trees in random forest*

n\_estimators = [int(x) for x in np.linspace(start = 10, stop = 300, 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]
```

```
# 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)
```

```
rf_Model = RandomForestClassifier()
```

CV can be changed below

```
from sklearn.model_selection import RandomizedSearchCV
```

```
rf_RandomGrid = RandomizedSearchCV(estimator = rf_Model, param_distributions = param_grid, cv = 10,
```

```
verbose=2, n_jobs = 4)
```

```
rf_RandomGrid.fit(X_train, y_train)
```

BEST PARAMETER RESULT

```
rf_RandomGrid.best_params_
```

## OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
from sklearn.metrics import RocCurveDisplay
```

```
import matplotlib.pyplot as plt
```

### ROC CURVE

```
rfc = RandomForestClassifier(n_estimators=10, random_state=42)
```

```
rfc.fit(X_train, y_train)
```

```
ax = plt.gca()
```

```
rfc_disp = RocCurveDisplay.from_estimator(rfc, X_test, y_test, ax=ax, alpha=0.2)
```

```
plt.show()
```

### COMPARING OTHER ALGORITHM (Naive Bayes Algorithm and Decision Tree)

#### Naive Bayes Algorithm Result

```
from sklearn.model_selection import train_test_split
```

```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=9) #Split the dataset
```

```
from sklearn.naive_bayes import GaussianNB
```

```
nv = GaussianNB() # create a classifier
```

```
nv.fit(X_train,y_train) # fitting the data
```

```
from sklearn.metrics import accuracy_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))
```

Logistic Regression

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.metrics import classification_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
```

RANDOM FOREST CLASSIFIER

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import classification_report
```

```
from sklearn.metrics import accuracy_score
```

```
rf=RandomForestClassifier(n_estimators=50,min_samples_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))
```

RidgeClassifier

```
from sklearn.linear_model import RidgeClassifier
```

```
from sklearn.datasets import load_iris
```

```
from sklearn.datasets import make_classification
```

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.model_selection import cross_val_score
```

```
from sklearn.metrics import confusion_matrix
```

```
from sklearn.metrics import classification_report
```

```
X, y = make_classification(n_samples=5000, n_features=10,
```

```
    n_classes=3,
```

```
    n_clusters_per_class=1)
```

```
xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.15)
```

```
rc = RidgeClassifier()
```

```
print(rc)
```

```
RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True, fit_intercept=True,
```

```
    max_iter=None, normalize=True, random_state=None, solver='auto',
```

```
    tol=0.001)
```

```
rc.fit(xtrain, ytrain)
```

```
score = rc.score(xtrain, ytrain)
```

```
print("Accuracy Score of RidgeClassifier is ", (score)*100)
```

Perception Model

```

from sklearn.linear_model import Perceptron

clf = Perceptron(tol=1e-3, random_state=0)

clf.fit(X, y)

Perceptron()

clf.score(X, y)

Perceptron()

print("Accuracy Score of Perceptron model is: ", (score)*100)

```

INPUT/TUNE VALUES OF INTEREST HERE (PARAMETER GRID)

```

In [13]: # Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 100, 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]

```

Fig: 4.2.15 INPUT/TUNE VALUES OF INTEREST ON PARAMETER GRID

```

In [14]: # 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)

{'n_estimators': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100], 'max_features': ['auto', 'sqrt'], 'max_depth': [2, 4], 'min_sam
ples_split': [2, 5], 'min_samples_leaf': [1, 2], 'criterion': ['gini', 'entropy'], 'bootstrap': [True, False]}

```

Fig: 4.2.16 RANDOM FOREST HYPERPARAMETER GRID DISPLAYED

```
Out[17]: 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, 20, 30, 40, 50, 60,
                                                            70, 80, 90, 100]},
                          verbose=2)
```

#### BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
```

```
Out[18]: {'n_estimators': 100,
          'min_samples_split': 2,
          'min_samples_leaf': 2,
          'max_features': 'sqrt',
          'max_depth': 4,
          'criterion': 'gini',
          'bootstrap': False}
```

Fig: 4.2.17 RANDOM FOREST BEST HYPERPARAMETER VALUES

#### BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
```

```
Out[18]: {'n_estimators': 100,
          'min_samples_split': 2,
          'min_samples_leaf': 2,
          'max_features': 'sqrt',
          'max_depth': 4,
          'criterion': 'gini',
          'bootstrap': False}
```

#### OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [19]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 98.101
```

Fig: 4.2.18 OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

## BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
```

```
Out[18]: {'n_estimators': 100,  
         'min_samples_split': 5,  
         'min_samples_leaf': 2,  
         'max_features': 'auto',  
         'max_depth': 4,  
         'criterion': 'gini',  
         'bootstrap': False}
```

## OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [19]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 98.146
```

Fig: 4.2.19 EXPERIMENT 15 ON FULL OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

## BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
```

```
Out[18]: {'n_estimators': 50,  
         'min_samples_split': 2,  
         'min_samples_leaf': 2,  
         'max_features': 'sqrt',  
         'max_depth': 4,  
         'criterion': 'gini',  
         'bootstrap': True}
```

## OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [19]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 97.881
```

Fig: 4.2.20 EXPERIMENT 16 ON FULL OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

## BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
Out[18]: {'n_estimators': 30,
          'min_samples_split': 5,
          'min_samples_leaf': 1,
          'max_features': 'sqrt',
          'max_depth': 4,
          'criterion': 'entropy',
          'bootstrap': False}
```

## OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [19]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 97.906
```

Fig: 4.2.21 EXPERIMENT 17 ON FULL OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

## BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
Out[18]: {'n_estimators': 20,
          'min_samples_split': 5,
          'min_samples_leaf': 1,
          'max_features': 'sqrt',
          'max_depth': 4,
          'criterion': 'entropy',
          'bootstrap': True}
```

## OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [19]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 97.624
```

Fig: 3.21 EXPERIMENT 17 ON FULL OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

## BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
```

```
Out[18]: {'n_estimators': 70,
          'min_samples_split': 5,
          'min_samples_leaf': 2,
          'max_features': 'auto',
          'max_depth': 4,
          'criterion': 'entropy',
          'bootstrap': False}
```

## OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [19]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 97.703
```

Fig: 4.2.22 EXPERIMENT 18 ON FULL OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

## BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
```

```
Out[18]: {'n_estimators': 26,
          'min_samples_split': 5,
          'min_samples_leaf': 2,
          'max_features': 'sqrt',
          'max_depth': 4,
          'criterion': 'gini',
          'bootstrap': True}
```

## OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [19]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 97.607
```

```
In [20]: from sklearn.metrics import RocCurveDisplay
```

Fig: 4.2.23 EXPERIMENT 19 ON FULL OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

## BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
```

```
Out[18]: {'n_estimators': 23,  
         'min_samples_split': 2,  
         'min_samples_leaf': 2,  
         'max_features': 'sqrt',  
         'max_depth': 4,  
         'criterion': 'entropy',  
         'bootstrap': False}
```

## OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [19]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 96.927
```

Fig: 4.2.24 EXPERIMENT 20 ON FULL OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

## BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
```

```
Out[18]: {'n_estimators': 42,  
         'min_samples_split': 5,  
         'min_samples_leaf': 1,  
         'max_features': 'sqrt',  
         'max_depth': 4,  
         'criterion': 'gini',  
         'bootstrap': True}
```

## OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [19]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 97.688
```

Fig: 4.2.25 EXPERIMENT 21 ON FULL OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

## BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
```

```
Out[18]: {'n_estimators': 14,  
         'min_samples_split': 5,  
         'min_samples_leaf': 1,  
         'max_features': 'auto',  
         'max_depth': 4,  
         'criterion': 'entropy',  
         'bootstrap': True}
```

## OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [19]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 97.971
```

Fig: 4.2.26 EXPERIMENT 22 ON FULL OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

## BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
```

```
Out[18]: {'n_estimators': 80,  
         'min_samples_split': 5,  
         'min_samples_leaf': 2,  
         'max_features': 'sqrt',  
         'max_depth': 4,  
         'criterion': 'entropy',  
         'bootstrap': False}
```

## OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [19]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 98.255
```

Fig: 4.2.27 EXPERIMENT 23 ON FULL OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

## BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
```

```
Out[18]: {'n_estimators': 64,  
         'min_samples_split': 2,  
         'min_samples_leaf': 2,  
         'max_features': 'auto',  
         'max_depth': 4,  
         'criterion': 'gini',  
         'bootstrap': True}
```

## OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [19]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 97.617
```

Fig: 4.2.28 EXPERIMENT 24 ON FULL OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

## BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
```

```
Out[18]: {'n_estimators': 72,  
         'min_samples_split': 5,  
         'min_samples_leaf': 2,  
         'max_features': 'sqrt',  
         'max_depth': 4,  
         'criterion': 'entropy',  
         'bootstrap': True}
```

## OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [19]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 98.156
```

Fig: 4.2.29 EXPERIMENT 25 ON FULL OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
```

```
Out[18]: {'n_estimators': 41,
          'min_samples_split': 2,
          'min_samples_leaf': 2,
          'max_features': 'sqrt',
          'max_depth': 4,
          'criterion': 'entropy',
          'bootstrap': True}
```

OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [19]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 97.412
```

Fig: 4.2.30 EXPERIMENT 26 ON FULL OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

BEST PARAMETER RESULT

```
In [44]: rf_RandomGrid.best_params_
```

```
Out[44]: {'n_estimators': 203,
          'min_samples_split': 5,
          'min_samples_leaf': 1,
          'max_features': 'auto',
          'max_depth': 4,
          'criterion': 'entropy',
          'bootstrap': True}
```

OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [45]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 98.215
```

Fig: 4.2.31 EXPERIMENT 27 ON FULL OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

## BEST PARAMETER RESULT

```
In [18]: rf_RandomGrid.best_params_
```

```
Out[18]: {'n_estimators': 203,
          'min_samples_split': 5,
          'min_samples_leaf': 1,
          'max_features': 'sqrt',
          'max_depth': 4,
          'criterion': 'entropy',
          'bootstrap': True}
```

## OPTIMISED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

```
In [19]: print (f'Train Accuracy - : {rf_RandomGrid.score(X_train,y_train)*100:.3f}')
```

```
Train Accuracy - : 98.153
```

Fig: 4.2.32 EXPERIMENT 28 ON FULL OPTIMIZED HYPERPARAMETER TUNNING OF RANDOM FOREST CLASSIFIER RESULT

## Logistic Regression

```
In [23]: import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_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
```

```
Out[23]: 89.2238813078993
```

Fig 4.2.33 COMPARISON WITH LOGISTIC REGRESSION ALGORITHM

## Naive Bayes Algorithm Result

```
In [22]: from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=9) #Split the dataset
from sklearn.naive_bayes import GaussianNB
nv = GaussianNB() # create a classifier
nv.fit(X_train,y_train) # fitting the data
from sklearn.metrics import accuracy_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
```

Fig 4.2.34 COMPARISON WITH NAÏVE BAYES ALGORITHM

## Perception Model

```
In [26]: from sklearn.linear_model import Perceptron
clf = Perceptron(tol=1e-3, random_state=0)
clf.fit(X, y)
Perceptron()
clf.score(X, y)
Perceptron()
print("Accuracy Score of Perceptron model is: ", (score)*100)

Accuracy Score of Perceptron model is: 89.52941176470588
```

Fig 4.2.35 COMPARISON PERCEPTION MODEL ALGORITHM

## RANDOM FOREST CLASSIFIER

```
In [24]: from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
rf=RandomForestClassifier(n_estimators=50,min_samples_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 : 91.82377455844414
```

Fig 4.2.36 COMPARISON WITH RANDOM FOREST ALGORITHM

## RidgeClassifier

```
In [25]: from sklearn.linear_model import RidgeClassifier
from sklearn.datasets import load_iris
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
X, y = make_classification(n_samples=5000, n_features=10,
                          n_classes=3,
                          n_clusters_per_class=1)

xtrain, xtest, ytrain, ytest = train_test_split(X, y, test_size=0.15)
rc = RidgeClassifier()
print(rc)

RidgeClassifier(alpha=1.0, class_weight=None, copy_X=True, fit_intercept=True,
                max_iter=None, normalize=True, random_state=None, solver='auto',
                tol=0.001)

rc.fit(xtrain, ytrain)
score = rc.score(xtrain, ytrain)
print("Accuracy Score of RidgeClassifier is ", (score)*100)

RidgeClassifier()
Accuracy Score of RidgeClassifier is 89.52941176470588
```

Fig 4.2.37 COMPARISON WITH RIDGE-CLASSIFIER ALGORITHM

## ROC CURVE

```
In [21]: rfc = RandomForestClassifier(n_estimators=10, random_state=42)
rfc.fit(X_train, y_train)
ax = plt.gca()
rfc_disp = RocCurveDisplay.from_estimator(rfc, X_test, y_test, ax=ax, alpha=0.2)
plt.show()
```

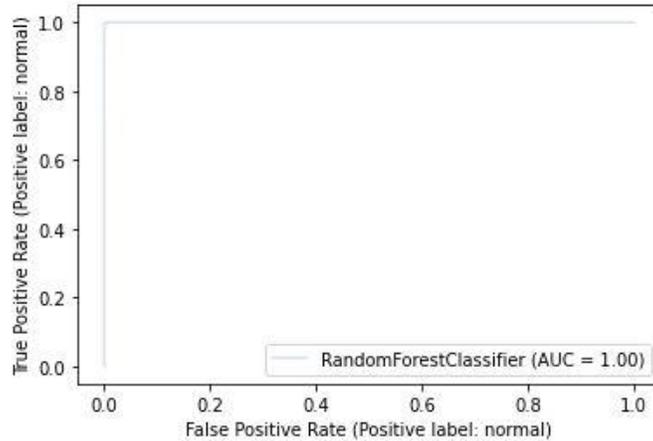


Fig 4.2.38 ROC (Receiver Operating Characteristic.) curve

S/N	NUMBER OF ESTIMATORS	MIN. SAMPLES_SPLIT	MIN SAMPLE_LEAF	MAX FEATURES	MAX. DEPTH	CRITE_RION	BOOT-STRAP	ACCURACY (%)
1	100	5	2	Auto	4	Gini	False	98.15
2	50	2	2	Sqrt	4	Gini	True	97.88
3	30	5	1	Sqrt	4	Entropy	False	97.91
4	20	5	1	Sqrt	4	Entropy	True	97.62
5	70	5	1	Sqrt	4	Entropy	False	97.70
6	26	5	2	Sqrt	4	Gini	True	97.61
7	23	2	2	Sqrt	4	Entropy	False	96.93
8	42	5	1	Sqrt	4	Entropy	False	97.69
9	14	5	1	Auto	4	Entropy	True	97.97
10	80	5	2	Sqrt	4	Entropy	False	98.26

11	64	2	2	Auto	4	Gini	True	97.62
12	72	5	2	Sqrt	4	Entropy	True	98.16
13	41	2	2	Sqrt	4	Entropy	True	97.41
14	203	5	1	Auto	4	Entropy	True	98.22

Table 4.2. Experimental summary on RandomizedCV full hyperparameter tuning.

## Discussion on Table 4.2

This section of experiment was carried out with full hyperparameter tuning which included the numbers of estimators, minimum sample leaf, min sample split, max features, max depth, criterion and bootstrap, results of the experiments clearly stated below the table. Experiments on using the RandomizedCV showed that when we had estimators values at 100, 80, 72, 203, the accuracies were 98.15%, 98.62%, 98.16% and 98.22% respectively. It was observed the performance in outcome with respect to accuracy was randomized, also on this experiment, it was observed that on some occasions, the greater the numbers of trees on the nodes, the more predictive the accuracy of the outcome. It was also observed from the randomizedSearchCV hyperparameter tuning that when the estimators value = 80, Min. Sample leaf = 2, min sample split = 5, max features =sqrt, max depth = 4, criterion = entropy, bootstrap = false, the efficiency on the outcome was 98.26% showing great adaptability of the model with respect to an increase numbers of trees on the nodes..

This is just a complete testing of my model to check for correctness on predictive purpose, and its performance was super.

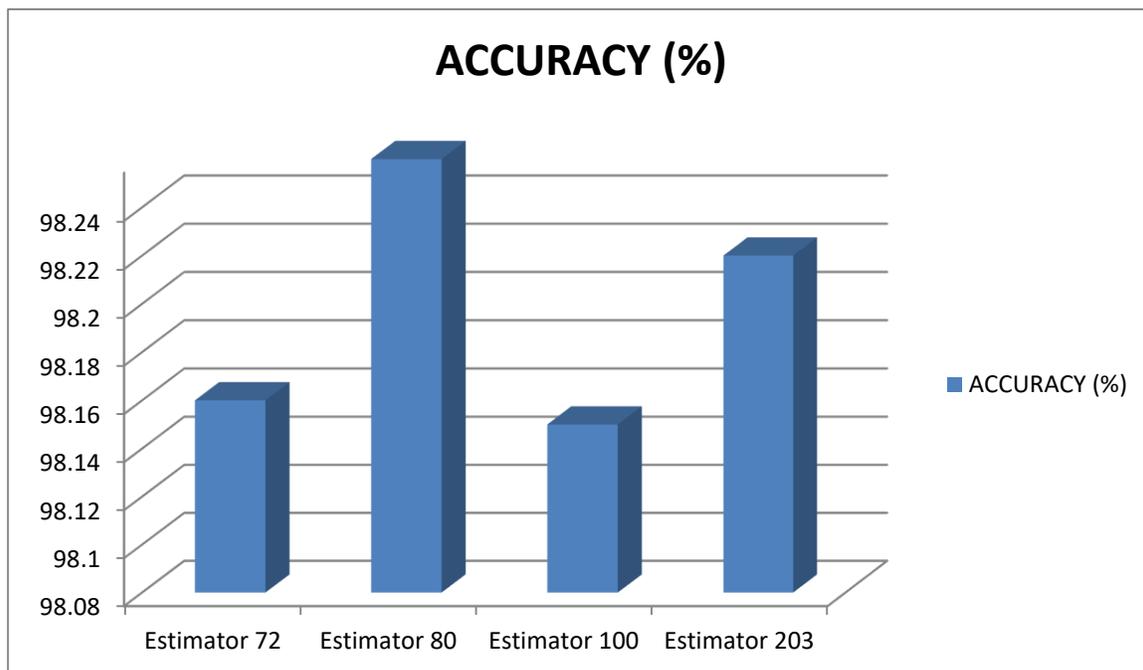


Fig 4.2.39 Graph depicting accuracy level of the RandomizedCV hyperparameter tuning.

ESTIMATORS	ACCURACY (%)
Estimator 72	98.16
Estimator 80	98.26
Estimator 100	98.15
Estimator 203	98.22

S/N	COMPARISON OF ALGORITHMIS	ACCURACY (%)
1.	NAÏVE BAYES	52.93
2.	RANDOM FOREST CLASSIFIER	91.82
3.	RIDGE CLASSIFIER	89.53
4.	PERCEPTION MODEL	89.52
5.	LOGISTICS REGRESSION	89.22
6	OPTIMIZED RANDOM FOREST CLASIFIER	98.26

Table 4.3 Experimental summary on Algorithms Comparison.

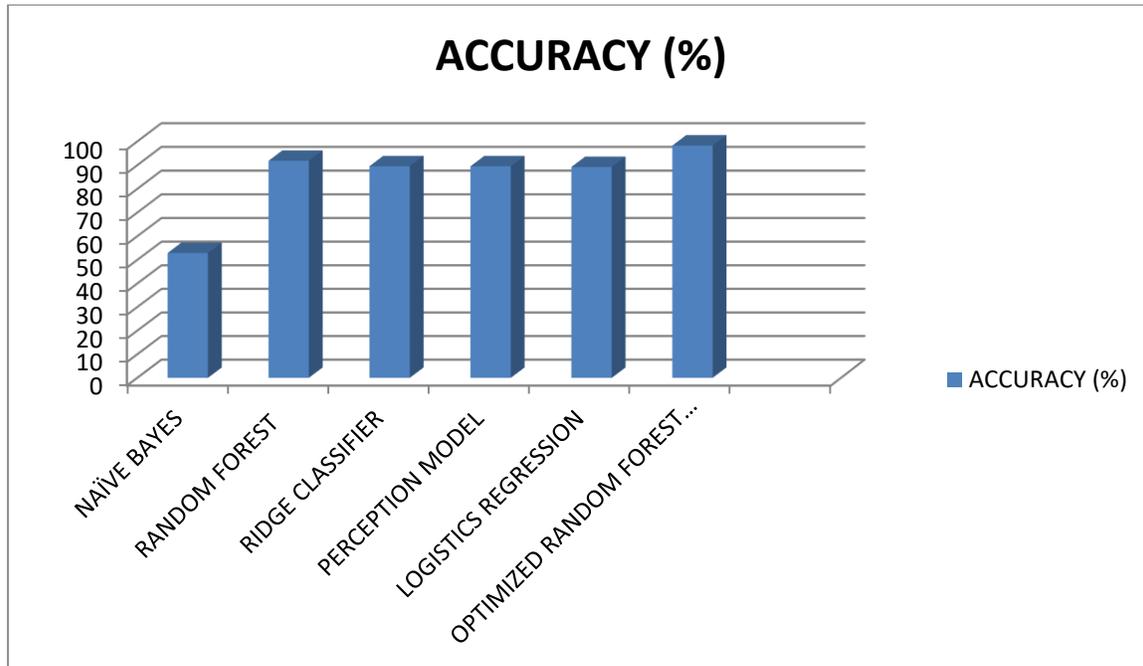


Fig 4..2.40 Graph depicting accuracy level of the comparison algorithm

## ROC CURVE

```
In [21]: rfc = RandomForestClassifier(n_estimators=10, random_state=42)
rfc.fit(X_train, y_train)
ax = plt.gca()
rfc_disp = RocCurveDisplay.from_estimator(rfc, X_test, y_test, ax=ax, alpha=0.2)
plt.show()
```

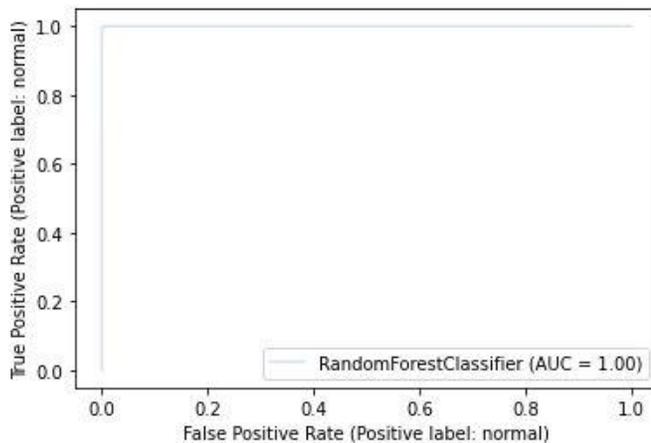


Fig ROC (Receiver Operating Characteristic.)

ROC (Receiver Operating Characteristic.) curve ROC curve is an evaluation metric that measures the performance of a machine learning model by visualizing accuracy is measured by the area under the ROC curve. An area of 1 represents a perfect test; an area of .5 represents a worthless test. From our ROC the area under (AU) the ROC is 1 which definitely represent a perfect test.

## SUMMARY

Below are the experimental inferences of this research work, to this end, various observational and computational analysis has contributed immensely to the successful completion of this work.

1). The X variables are the feature variables while the Y variable is the target variable.

X is also refers to as independent variables while Y is known as the dependent variable.

2). X\_train - This includes all the independent variables that will be used to train the model, also as we have specified the test\_size = 0.2, this means 80% of observations from the complete data will be used to train/fit the model and rest 20% will be used to test the model.

3). X\_test – The remaining 20% portion of the independent variables from the data which is not not to be used in the training phase but rather, it will be used to make predictions to test the accuracy of the model.

4). Y\_train - This is the dependent variable which needs to be predicted by this model, this includes category labels against the independent variables, we need to specify our dependent variable while training/fitting the model.

5). Y\_test - This data has category labels for your test data, these labels will be used to test the accuracy between actual and predicted categories.

All the above mentioned variables are specified accordingly in our project code

6). Comparison analysis with other algorithm ( added to the optimized model)

## FINDINGS OF STUDY

The model consist of the properties of decision trees, this combination gives Random forest algorithm a better performance. After the execution of the program, the best Optimized Random Forest behavioural hyper-parameters tuning result yielded has 98.26%, approximately 100% while Naïve Bayes has 52.93% and Ridge Classifier had 89.53% while our default Random Forest classifier has 91.82 %.\

## CONCLUSIONS

This proves that the Optimized Random Forest hyperparameter tuning is more efficient and better in terms of practical usage than that of the default Random forest algorithms and some other machine learning algorithms, its reliability also encourages future predictions to be forecasted which goes a long way to solving machine learning problems.

## References

- (1) Barreiros M, Lundqvist P (2011). QoS-Enabled Networks: Tools and Foundations. West Sussex, UK: John Wiley & Sons.
- (2) Bhojar, 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) Koendjibiharie 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 Baselineing (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) 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 Amaku Amaku, 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) Amaku Amaku, 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) Amaku Amaku, Rapheal Watti, Igbinosa 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. Muoz Castaeda, D. Escudero Garca, 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. B314