CSE 3501

Information Security Analysis and Audit

J-Component Report

IOT-Network Intrusion Detection Using ML Stacking Algorithm

FALL SEMESTER 2022-23

Submitted By:

Anish Desai 20BCE0461

Aditya Krishna 20BCE0456

Vasu Garg 20BCE0451

Guided By:

Prof. Lavanya K.

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Table of Contents

I. Brief Introduction

1.1 Context

Increased Cyber-attacks in network layers and need for effective solution that uses futuristic technologies such as Machine Learning.

1.2 Problem Statement

With increasing dependency on the network, malicious attacks on the network layers, especially in the data packets layer has increased drastically. Mostly these activities include inclusion of malicious data packets along with normal data packets, which can compromise the entire network security. Therefore, in order to protect the network from these malicious packets, it is required to identify if a packet is malicious or not.

1.3 Tech-Stack Used

Python 3.9
Py Lib - Sklearn, Seaborn, Pandas, Numpy
IDE - Jupyter Notebook
EDA and Machine Learning Models

1.4 Dataset

The KDDCup99 is the original IoT network intrusion dataset that was created in 1999. The motivation behind the dataset's creation was to improve the capability of IDSs.

The dataset used is the latest version of NSL-KDD, the data for which has been gathered from Massachusetts Institute of Technology - Lincoln labs and simulates a typical US Air Force Local Area Network (IOT-LAN).

https://www.kaggle.com/datasets/airadix/nslkdd?select=NSL_KDD_Test.csv

Other IOT-network-based datasets can be used. Prominent among them is UNSW_NB15.csv dataset. The BoT-IoT dataset was created by designing a realistic network environment in the Cyber Range Lab of UNSW Canberra. The network environment incorporated a combination of normal and botnet traffic. The dataset's source files are provided in different formats, including the

original pcap files, the generated argus files and csv files. The files were separated, based on attack category and subcategory, to better assist in labeling process.

1.5 Methodology

Assessing the performance of basic classification ML models with respect to detecting network intrusions and then stacking them to frame a Stacking Classifier Model.

Combining Stacking with Bagging and Boosting Methods which together constitute the ENTIRE Ensemble Learning Model of ML.

1.6 Performance Metrics

Accuracy for each classification model.

Confusion matrix depicting the number of TPs, TNs, FPs, and FNs as classified by each model.

Classification report presenting various parameters of performance such as Precision, Recall and F-score.

II. Explanation of Models/Algorithms

2.1 Models Used

Traditional Classification Models & its Stacking

- Logistic Regression
- KNN Algorithm
- CART Decision Tree Algorithm
- Support Vector Machine
- Naive Bayesian Model

Ensemble Learning Methods & its Stacking

- XGBoost Method
- Random Forest Method
- Adaboost Method

2.2 NSL-KDD Dataset

The NSL-KDD dataset has a total of 25,192 data packet samples of the IOT network. It has a total of 41 features for each data packet that constitute the deciding factors for classification and detection. The features consists of source IP address, destination IP address and port numbers, protocol used, server rate, and duration, etc.

```
In [2]: #Importing the dataset
        df=pd.read_csv("nsl_kdd.csv")
        df.head()
```

Out[2]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_d
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	0.17	
1	0	udp	other	SF	146	0	0	0	0	0	...	1	0.00	
2	0	tcp	private	S0	0	0	0	0	0	0	...	26	0.10	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	1.00	
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	1.00	

5 rows × 42 columns

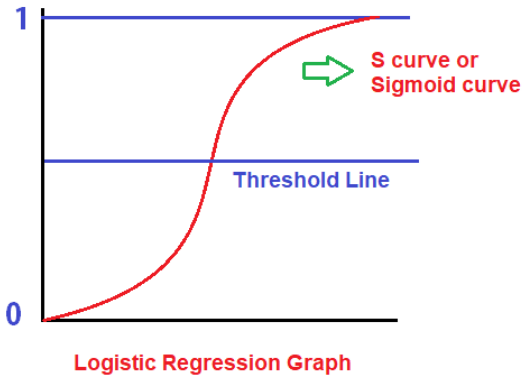
```
In [3]: #Dataset Size
        df.shape
```

```
Out[3]: (25192, 42)
```

2.3 Logistic Regression

Logistic Regression is the first traditional supervised classification model used for analysis. It is used for predicting the categorical dependent variable using a given set of independent variables.

If the predicted probability of the class is more than the threshold value, the predicted class value will be 1 or else 0. This gives rise to a sigmoid graph.

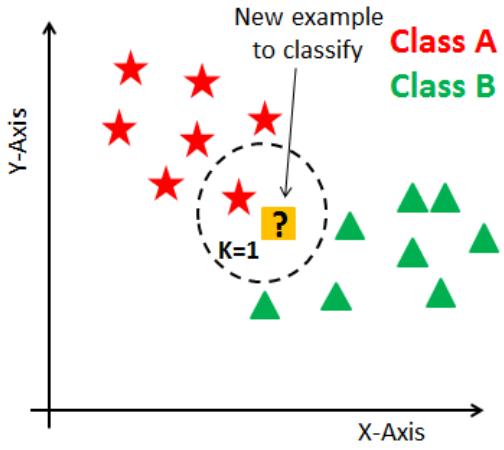


Logistic Regression Graph

In our demonstration, 0 indicates that the data packet is malicious and 1 indicates a normal data packet. The threshold value is generally set at 0.5.

2.4 KNN Algorithm

K-nearest neighbors' algorithm is the second traditional supervised algorithm used in our analysis. It finds the k-nearest neighbors (k initialized) to the testing data point and maps the test datapoint to the majority or the aggregate average of the k-nearest neighbors.



KNN is used for both classification as well as regression problems.

In case of classification, we choose class label of the majority of k-neighbors. In case of regression, we calculate the total aggregate of the k-neighbors and compute the average value. This is the class label value of the test data sample.

2.5 CART Decision Tree Algorithm

The CART algorithm is a type of classification algorithm that is required to build a decision tree on the basis of Gini's impurity index.

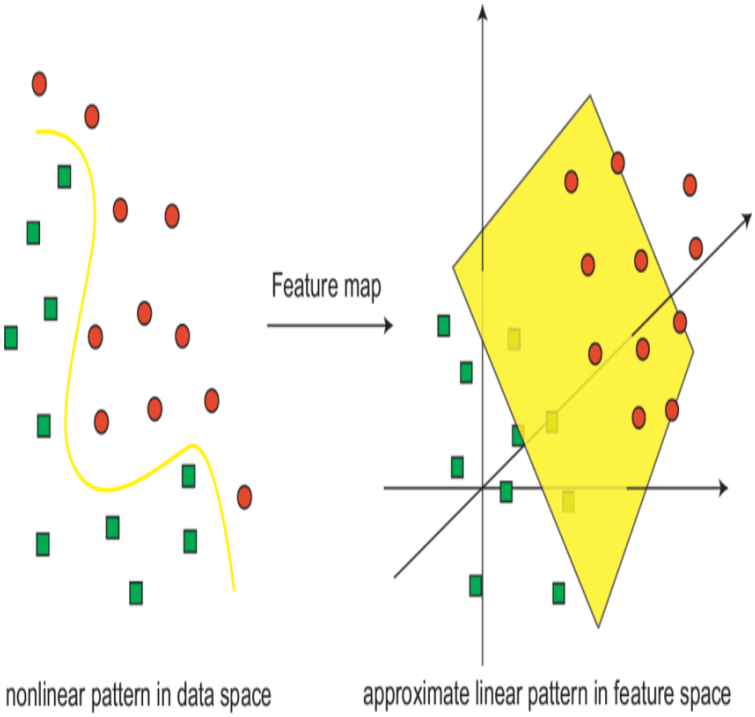
$$Gini(split) = \sum_{i=1}^{n} p_i * (1 - p_i).$$

The nodes are split into sub-nodes on the basis of a threshold value of an attribute. The CART algorithm does that by searching for the best homogeneity for the sub-nodes, with the help of the Gini Index criterion.

The root node is taken as the training set and is split into two by considering the best attribute and threshold value. Further, the subsets are also split using the same logic. This continues till the last pure sub-set is found in the tree or the maximum number of leaves possible in that growing tree.

2.6 Support Vector Machine

SVM finds a hyper-plane that creates a boundary between the types of data.

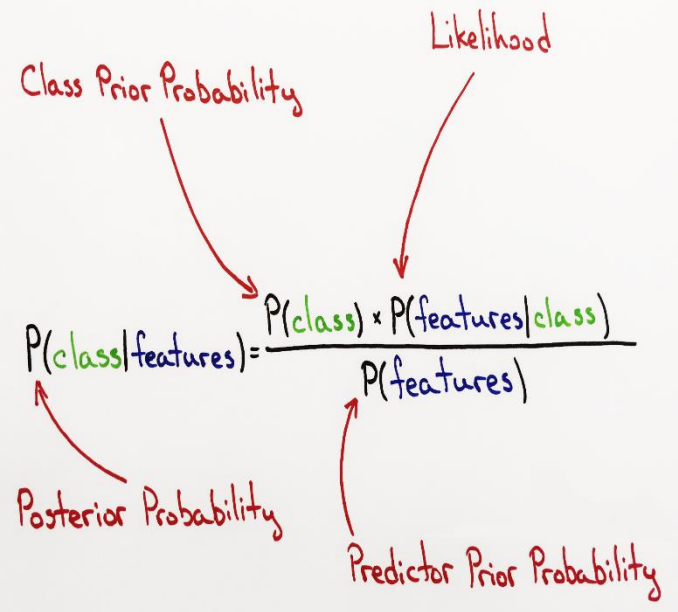


In SVM, we plot each data item in the dataset in an N-dimensional space, where N is the number of features/attributes in the data. Next, find the optimal hyperplane to separate the data. So by this, you must have understood that inherently, SVM can only perform binary classification (i.e., choose between two classes)

2.7 Naïve Bayesian Model

Naïve Bayes is a probabilistic machine learning algorithm used for many classification functions and is based on the Bayes theorem.

Calculates the probability of the test data sample with respect to both the class labels.

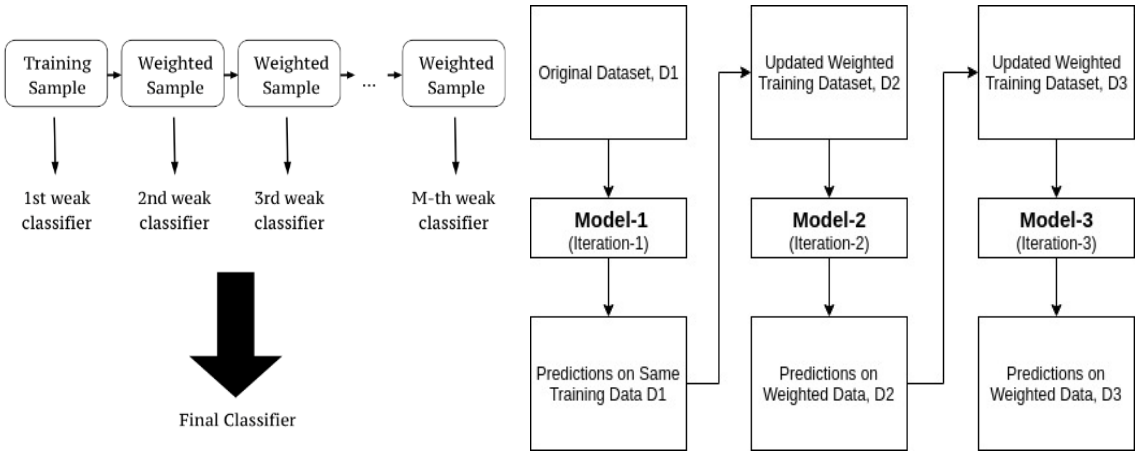


The class label with higher probability is the resultant class label.

2.8 XGBoost Method

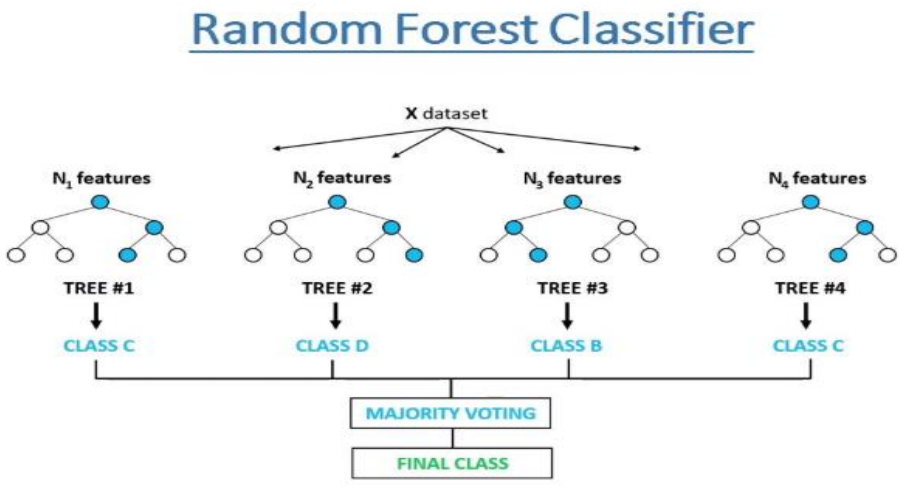
Abbreviation for Extreme Gradient Boosting.

In this algorithm, decision trees are created in sequential form. Weights play an important role in XGBoost. Weights are assigned to all the independent variables which are then fed into the decision tree which predicts results. The weight of variables predicted wrong by the tree is increased and these variables are then fed to the second decision tree. These individual classifiers/predictors then ensemble to give a strong and more precise model.



2.9 Random Forest Method

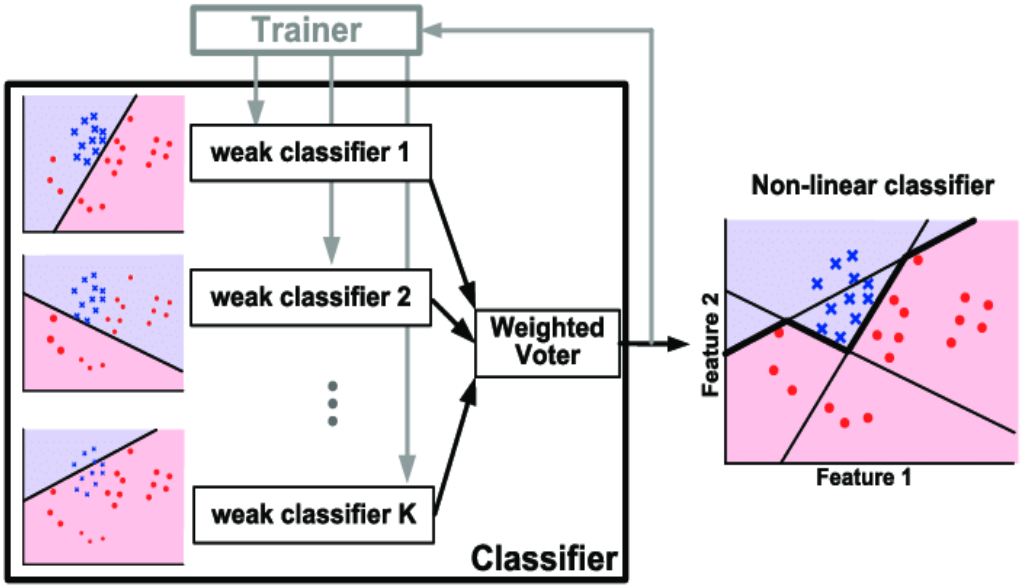
Random forest classifier creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object.



It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

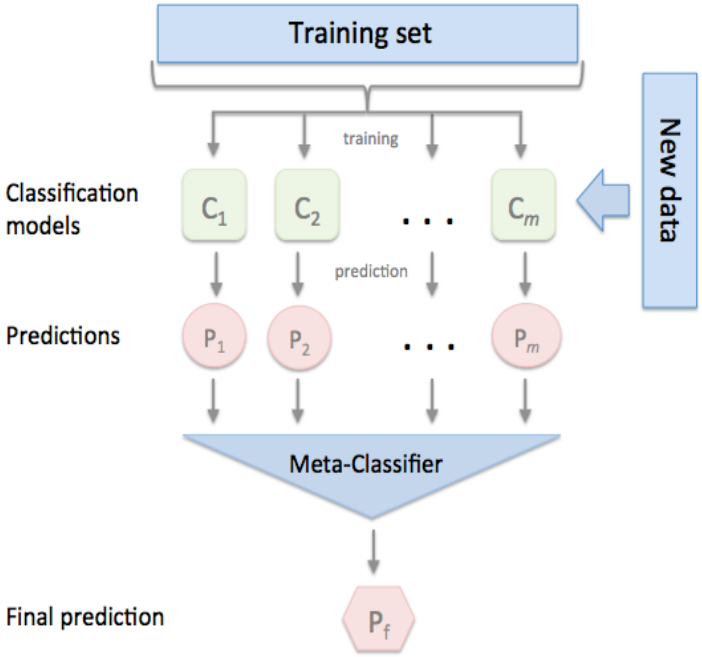
2.10 AdaBoost Method

AdaBoost stands for Adaptive Boosting where a set of simple and weak classifiers is used to achieve an improved classifier by emphasizing the samples that are misclassified by weak classifiers.



AdaBoost uses the complete training dataset to train the weak learners, where the training examples are reweighted in each iteration to build a strong classifier that learns from the mistakes of the previous weak learners in the ensemble.

2.11 Stacking Classifier : The final ultimate detector



The first layer has 'm' number of classification models stacked over each other. Each classification model makes its own predictions which is then passed onto the next meta-classifier layer. The meta-classifier then makes the final predictions and constitutes the stacking classifier model prediction.

III. Implementation of Stacking Classifier Model – 1

3.1 Importing Required Libraries and the Dataset

In [1]:
```
#Importing Required Libraries
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

In [2]:
```
#Importing the dataset
df=pd.read_csv("nsl_kdd.csv")
df.head()
```

Out[2]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_d
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	0.17	
1	0	udp	other	SF	146	0	0	0	0	0	...	1	0.00	
2	0	tcp	private	S0	0	0	0	0	0	0	...	26	0.10	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	1.00	
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	1.00	

5 rows × 42 columns

3.2 Exploratory Data Analysis (EDA)

In [3]:
```
#Dataset Size
df.shape
```

Out[3]: (25192, 42)

In [4]:
```
#Stats
df.describe()
```

Out[4]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromise
count	25192.000000	2.519200e+04	2.519200e+04	25192.000000	25192.000000	25192.00000	25192.000000	25192.000000	25192.000000	25192.0000
mean	305.054104	2.433063e+04	3.491847e+03	0.000079	0.023738	0.00004	0.198039	0.001191	0.394768	0.2278
std	2686.555640	2.410805e+06	8.883072e+04	0.008910	0.260221	0.00630	2.154202	0.045418	0.488811	10.4173
min	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.0000
25%	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.0000
50%	0.000000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.0000
75%	0.000000	2.790000e+02	5.302500e+02	0.000000	0.000000	0.00000	0.000000	0.000000	1.000000	0.0000
max	42862.000000	3.817091e+08	5.151385e+06	1.000000	3.000000	1.00000	77.000000	4.000000	1.000000	884.0000

8 rows × 38 columns

3.3 Label Encoding

In [5]:
```
#Label Encoding the non-numeric features
from sklearn.preprocessing import LabelEncoder
labelen = LabelEncoder()
df['protocol_type'] = labelen.fit_transform(df['protocol_type'])
df['service'] = labelen.fit_transform(df['service'])
df['flag'] = labelen.fit_transform(df['flag'])
df['class'] = labelen.fit_transform(df['class'])
```

In [6]:
```
#Label-encoded dataset
df.head()
```

Out[6]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_di
0	0	1	19	9	491	0	0	0	0	0	...	25	0.17	
1	0	2	41	9	146	0	0	0	0	0	...	1	0.00	
2	0	1	46	5	0	0	0	0	0	0	...	26	0.10	
3	0	1	22	9	232	8153	0	0	0	0	...	255	1.00	
4	0	1	22	9	199	420	0	0	0	0	...	255	1.00	

5 rows × 42 columns

3.4 Features and Class Label Datasets

In [7]:
```
#Splitting the dataset into
#features and class label
X=df.iloc[:,0:41]
y=df.iloc[:,41:42]
```

In [8]:
```
#Features dataset
X.shape
```

Out[8]: (25192, 41)

In [9]:
```
#Class Label data
y.shape
```

Out[9]: (25192, 1)

3.5 Training and Testing Datasets

In [10]:
```
#Splitting the features dataset into
#70% training dataset and 30% testing dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=10)
```

3.6 Logistic Regression

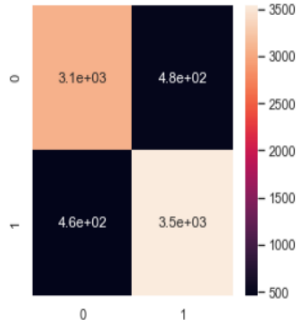
In [18]:
```
lr=LogisticRegression()
lr.fit(X_train,y_train)
y_pred_lr = lr.predict(X_test)
accuracy_lr=accuracy_score(y_test,y_pred_lr)
print("Accuracy: %.2f%%" % (accuracy_lr * 100.0))
```

```
C:\Users\ANISHDESAI\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was pa
ssed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

```
Accuracy: 87.58%
```

In [19]:
```
sns.set(rc={'figure.figsize':(4,4)})
cm = confusion_matrix(y_test,y_pred_lr)
sns.heatmap(cm,annot=True)
```

Out[19]: <AxesSubplot:>



3.7 KNN Algorithm

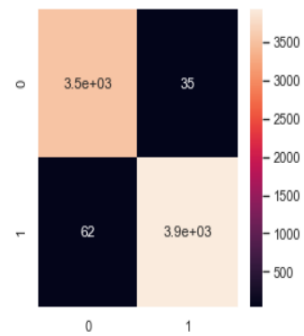
In [20]:
```
knn=KNeighborsClassifier()
knn.fit(X_train,y_train)
y_pred_knn = knn.predict(X_test)
accuracy_knn=accuracy_score(y_test,y_pred_knn)
print("Accuracy: %.2f%%" % (accuracy_knn * 100.0))
```

```
C:\Users\ANISHDESAI\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:179: DataConversionWarning: A column-vecto
r y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
```

```
Accuracy: 98.72%
```

In [21]:
```
sns.set(rc={'figure.figsize':(4,4)})
cm = confusion_matrix(y_test,y_pred_knn)
sns.heatmap(cm,annot=True)
```

Out[21]: <AxesSubplot:>



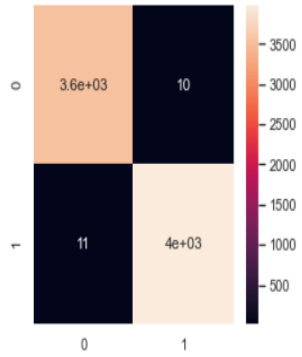
3.8 CART Decision Tree Algorithm

In [22]:
```
cart=DecisionTreeClassifier()
cart.fit(X_train,y_train)
y_pred_cart = cart.predict(X_test)
accuracy_cart=accuracy_score(y_test,y_pred_cart)
print("Accuracy: %.2f%%" % (accuracy_cart * 100.0))
```

```
Accuracy: 99.72%
```

In [23]:
```
sns.set(rc={'figure.figsize':(4,4)})
cm = confusion_matrix(y_test,y_pred_cart)
sns.heatmap(cm,annot=True)
```

Out[23]: <AxesSubplot:>



3.9 Support Vector Machine

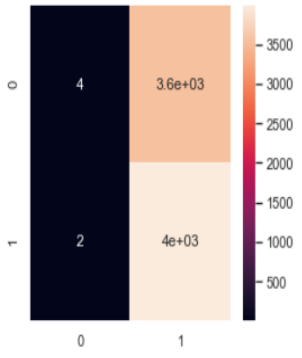
In [24]:
```
svm=SVC()
svm.fit(X_train,y_train)
y_pred_svm = svm.predict(X_test)
accuracy_svm=accuracy_score(y_test,y_pred_svm)
print("Accuracy: %.2f%%" % (accuracy_svm * 100.0))
```

```
C:\Users\ANISHDESAI\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was pa
ssed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

```
Accuracy: 52.91%
```

In [25]:
```
sns.set(rc={'figure.figsize':(4,4)})
cm = confusion_matrix(y_test,y_pred_svm)
sns.heatmap(cm,annot=True)
```

Out[25]: <AxesSubplot:>



3.10 Gaussian Naïve Bayesian Model

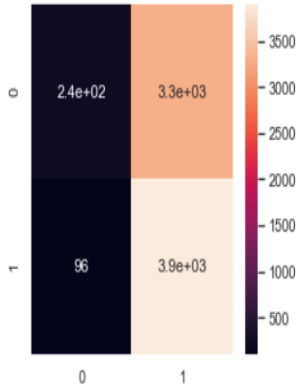
In [26]:
```
nb=GaussianNB()
nb.fit(X_train,y_train)
y_pred_nb = nb.predict(X_test)
accuracy_nb=accuracy_score(y_test,y_pred_nb)
print("Accuracy: %.2f%%" % (accuracy_nb * 100.0))
```

```
Accuracy: 54.83%
```

```
C:\Users\ANISHDESAI\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was pa
ssed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

In [27]:
```
sns.set(rc={'figure.figsize':(4,4)})
cm = confusion_matrix(y_test,y_pred_nb)
sns.heatmap(cm,annot=True)
```

Out[27]: <AxesSubplot:>



3.11 The StackingClassifier

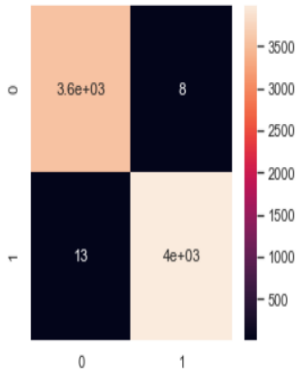
In [29]:
```
from sklearn.ensemble import StackingClassifier
#Define the base models
level0 = list()
level0.append(('lr', LogisticRegression()))
level0.append(('knn', KNeighborsClassifier()))
level0.append(('cart', DecisionTreeClassifier()))
level0.append(('svm', SVC()))
level0.append(('bayes', GaussianNB()))
#Define meta learner model
level1 = LogisticRegression()
#Define the stacking ensemble
model = StackingClassifier(estimators=level0, final_estimator=level1, cv=5)
```

In [30]:
```
model.fit(X_train,y_train)
y_pred_model = model.predict(X_test)
accuracy_model=accuracy_score(y_test,y_pred_model)
print("Accuracy: %.2f%%" % (accuracy_model * 100.0))
```

```
Accuracy: 99.72%
```

```
In [31]: sns.set(rc={'figure.figsize':(4,4)})
         cm = confusion_matrix(y_test,y_pred_model)
         sns.heatmap(cm,annot=True)
```

```
Out[31]: <AxesSubplot:>
```



In the Stacking Classifier Model – 1, we have stacked the five traditional classification models: LR, KNN, CART, SVM & GaussianNB in the first layer of the Stacking Classifier. The predictions are obtained and passed onto the Meta-classifier layer of the Stacking Classifier. This layer is modelled by Logistic Regression Algorithm (infact, can be modelled by any of the classification algorithm), so as to give maximum performance.

The individual accuracies obtained range from as low as ~52% to as high as ~99.7%. The Stacking Classifier combines the efficiencies of all the models and uses meta-classifier technique to give the best accuracy of 99.72%.

As obtained in the Confusion Matrix, the number of False Negatives and False Positives as predicted by our Stacking Classifier are very negligible (8 and 13 respectively) as compared to the True Positives and True Negatives.

IV. Implementation of Stacking Classifier Model – 2

4.1 Importing Required Libraries and the Dataset

In [1]:
```
#Importing Required Libraries
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import seaborn as sns
from sklearn.metrics import confusion_matrix
```

In [2]:
```
#Importing the dataset
df=pd.read_csv("nsl_kdd.csv")
df.head()
```

Out[2]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_d
0	0	tcp	ftp_data	SF	491	0	0	0	0	0	...	25	0.17	
1	0	udp	other	SF	146	0	0	0	0	0	...	1	0.00	
2	0	tcp	private	S0	0	0	0	0	0	0	...	26	0.10	
3	0	tcp	http	SF	232	8153	0	0	0	0	...	255	1.00	
4	0	tcp	http	SF	199	420	0	0	0	0	...	255	1.00	

5 rows × 42 columns

4.2 Exploratory Data Analysis

In [3]:
```
#Dataset Size
df.shape
```

Out[3]: (25192, 42)

In [4]:
```
#Stats
df.describe()
```

Out[4]:

	duration	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	num_failed_logins	logged_in	num_compromise
count	25192.000000	2.519200e+04	2.519200e+04	25192.000000	25192.000000	25192.00000	25192.000000	25192.000000	25192.000000	25192.0000
mean	305.054104	2.433063e+04	3.491847e+03	0.000079	0.023738	0.00004	0.198039	0.001191	0.394768	0.2278
std	2686.555640	2.410805e+06	8.883072e+04	0.008910	0.260221	0.00630	2.154202	0.045418	0.488811	10.4173
min	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.0000
25%	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.0000
50%	0.000000	4.400000e+01	0.000000e+00	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.0000
75%	0.000000	2.790000e+02	5.302500e+02	0.000000	0.000000	0.00000	0.000000	0.000000	1.000000	0.0000
max	42862.000000	3.817091e+08	5.151385e+06	1.000000	3.000000	1.00000	77.000000	4.000000	1.000000	884.0000

8 rows × 38 columns

4.3 Label Encoding

In [5]:
```
#Label Encoding the non-numeric features
from sklearn.preprocessing import LabelEncoder
labelen = LabelEncoder()
df['protocol_type'] = labelen.fit_transform(df['protocol_type'])
df['service'] = labelen.fit_transform(df['service'])
df['flag'] = labelen.fit_transform(df['flag'])
df['class'] = labelen.fit_transform(df['class'])
```

In [6]:
```
#Label-encoded dataset
df.head()
```

Out[6]:

	duration	protocol_type	service	flag	src_bytes	dst_bytes	land	wrong_fragment	urgent	hot	...	dst_host_srv_count	dst_host_same_srv_rate	dst_host_di
0	0	1	19	9	491	0	0	0	0	0	...	25	0.17	
1	0	2	41	9	146	0	0	0	0	0	...	1	0.00	
2	0	1	46	5	0	0	0	0	0	0	...	26	0.10	
3	0	1	22	9	232	8153	0	0	0	0	...	255	1.00	
4	0	1	22	9	199	420	0	0	0	0	...	255	1.00	

5 rows × 42 columns

4.4 Splitting Dataset

In [7]:
```
#Splitting the dataset into
#features and class label
X=df.iloc[:,0:41]
y=df.iloc[:,41:42]
```

In [8]:
```
#Features dataset
X.shape
```

Out[8]: (25192, 41)

In [9]:
```
#Class Label data
y.shape
```

Out[9]: (25192, 1)

In [10]:
```
#Splitting the features dataset into
#70% training dataset and 30% testing dataset
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=10)
```

4.5 XGBoost Method

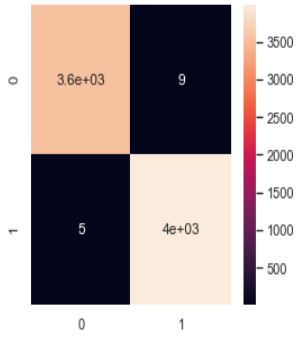
In [11]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier(n_estimators=200, learning_rate=0.6, max_depth=5, random_state=10)
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)
accuracy_xgb=accuracy_score(y_test,y_pred_xgb)
print("Accuracy: %.2f%%" % (accuracy_xgb * 100.0))
```

```
C:\Users\ANISHDESAI\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in XGBClassifier
is deprecated and will be removed in a future release. To remove this warning, do the following: 1) Pass option use_label_encod
er=False when constructing XGBClassifier object; and 2) Encode your labels (y) as integers starting with 0, i.e. 0, 1, 2, ...,
[num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
C:\Users\ANISHDESAI\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was pa
ssed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

```
[19:01:44] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
Accuracy: 99.81%
```

In [12]:
```
sns.set(rc={'figure.figsize':(4,4)})
cm = confusion_matrix(y_test,y_pred_xgb)
sns.heatmap(cm,annot=True)
```

Out[12]: <AxesSubplot:>



4.6 Random Forest Method

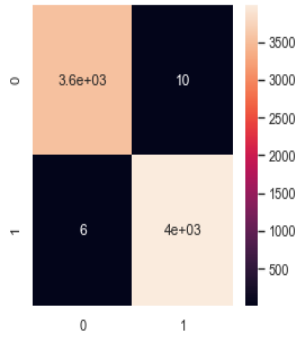
In [13]:
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(random_state=1)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
accuracy_rf=accuracy_score(y_test,y_pred_rf)
print("Accuracy: %.2f%%" % (accuracy_rf * 100.0))
```

```
<ipython-input-13-a5fc2db5b7b9>:3: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please cha
nge the shape of y to (n_samples,), for example using ravel().
  rf.fit(X_train, y_train)
```

```
Accuracy: 99.79%
```

In [14]:
```
sns.set(rc={'figure.figsize':(4,4)})
cm = confusion_matrix(y_test,y_pred_rf)
sns.heatmap(cm,annot=True)
```

Out[14]: <AxesSubplot:>



4.7 AdaBoost Method

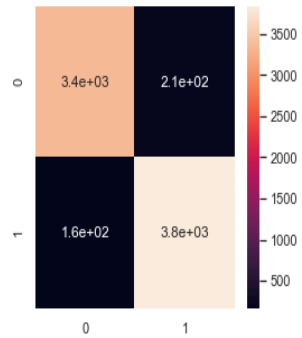
In [15]:
```
from sklearn.ensemble import AdaBoostClassifier
abc = AdaBoostClassifier(n_estimators=5)
abc.fit(X_train, y_train)
y_pred_abc = abc.predict(X_test)
accuracy_abc=accuracy_score(y_test,y_pred_abc)
print("Accuracy: %.2f%%" % (accuracy_abc * 100.0))
```

```
Accuracy: 95.17%
```

```
C:\Users\ANISHDESAI\anaconda3\lib\site-packages\sklearn\utils\validation.py:63: DataConversionWarning: A column-vector y was pa
ssed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  return f(*args, **kwargs)
```

In [16]:
```
sns.set(rc={'figure.figsize':(4,4)})
cm = confusion_matrix(y_test,y_pred_abc)
sns.heatmap(cm,annot=True)
```

Out[16]: <AxesSubplot:>



4.8 The StackingClassifier

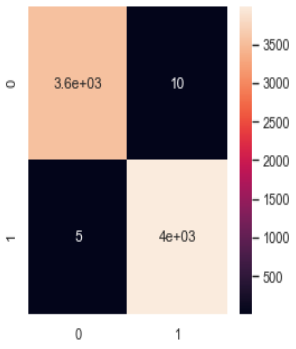
In [17]:
```
from sklearn.ensemble import StackingClassifier
#Define the base models
level0 = list()
level0.append(('xgb', XGBClassifier()))
level0.append(('rf', RandomForestClassifier()))
level0.append(('abc', AdaBoostClassifier()))
#Define meta learner model
level1 = LogisticRegression()
#Define the stacking ensemble
model = StackingClassifier(estimators=level0, final_estimator=level1, cv=5)
```

In [18]:
```
model.fit(X_train,y_train)
y_pred_model = model.predict(X_test)
accuracy_model=accuracy_score(y_test,y_pred_model)
print("Accuracy: %.2f%%" % (accuracy_model * 100.0))
```

```
[19:02:53] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
Accuracy: 99.80%
```

In [19]:
```
sns.set(rc={'figure.figsize':(4,4)})
cm = confusion_matrix(y_test,y_pred_model)
sns.heatmap(cm,annot=True)
```

Out[19]: <AxesSubplot:>



In [20]:
```
#Performance Metrics of Model
from sklearn.metrics import classification_report
print('Classification Report : \n')
print(classification_report(y_test,y_pred_model))
```

```
Classification Report :

              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3561
           1       1.00      1.00      1.00      3997

    accuracy                           1.00      7558
   macro avg       1.00      1.00      1.00      7558
weighted avg       1.00      1.00      1.00      7558
```

In the Stacking Classifier Model – 2, we have stacked the three ensemble classification models: XGBoost, Random Forest & AdaBoost in the first layer of the Stacking Classifier. The predictions are obtained and passed onto the Meta-classifier layer of the Stacking Classifier. This layer is modelled by

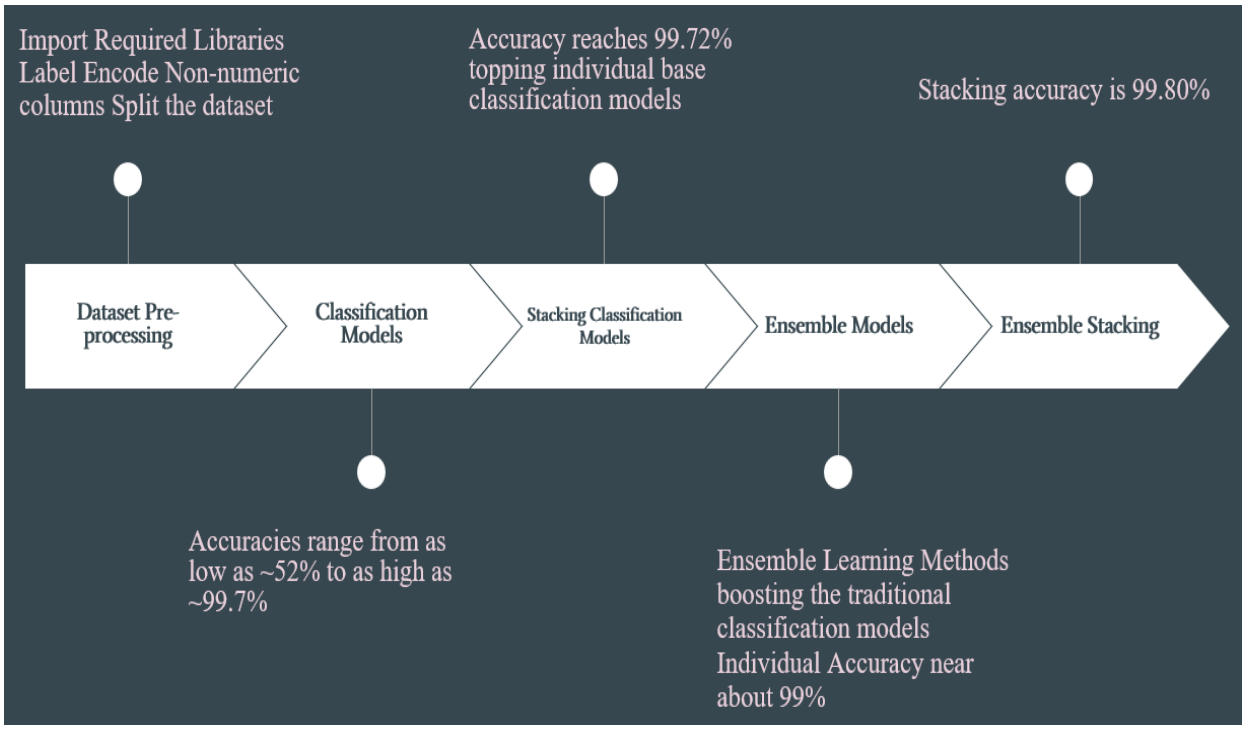
Logistic Regression Algorithm (infact, can be modelled by any of the classification algorithm), so as to give maximum performance.

The individual accuracies obtained are around 99%. The Stacking Classifier combines the efficiencies of all the models and uses meta-classifier technique to give the best accuracy of 99.80%.

As obtained in the Confusion Matrix, the number of False Negatives and False Positives as predicted by our Stacking Classifier are very negligible (10 and 5 respectively) as compared to the True Positives and True Negatives.

The Classification Report Analysis obtained shows the Precision, Recall and F-score measures to be a perfect 1.0. This signifies that the model devised can be used for an effective and efficient Intrusion Detection.

V. Summary



Glimpse of our methodology and results

VI. Conclusion

We can infer from the above results that StackingClassifier improves the detection performance of traditional classification models used for intrusion detection.

We can also infer that StackingClassifier when used with the advanced versions of the traditional classification models i.e., the ensemble methods, they give near-about same accuracy and precision as the individual components.

This makes essentially a StackingClassifier an equivalent of the Ensemble Learning Methods – Bagging and Boosting with respect to IOT-network Intrusion Detection.

----------The End----------