Group 10

Sudhay Senthilkumar, Sandeep Rajakrishnan

Amrita School of Engineering - Coimbatore

Botnet Threat Detection using Machine Learning Techniques

Sandeep Rajakrishnan, Sudhay Senthilkumar,
Senthilkumar Thangavel[*1] and Sulakshan Vajipayajula[*2]
Department of Computer Science and Engineering
[*1]Amrita School of Engineering, Coimbatore,
Amrita Vishwa Vidyapeetham, India.
[*2]Architect-CTO Office
IBM Security, Bangalore

sandur43@gmail.com ,
sudhay2001@gmail.com ,
t_senthilkumar@cb.amrita.edu [*1],
svajipay@in.ibm.com [*2]

```
pip install category_encoders
```

```
Requirement already satisfied: category_encoders in /usr/local/lib/python3.7/dist-packages (2.3.0)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (0.10.2)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (0.5.2)
Requirement already satisfied: pandas>=0.21.1 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (1.1.5)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (1.19.5)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (1.0.1)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from category_encoders) (1.4.1)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.21.1->category_encoders) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.21.1->category_encoders)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from patsy>=0.5.1->category_encoders) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20.0->category_encode
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=0.20.0->category_encoders) (1.1
```

```
import pandas as pd
import numpy as np
import category_encoders as ce
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
```

```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
```

```
Mounted at /content/drive
```

```
data=pd.read_csv(r"/content/drive/MyDrive/iot_botnet_dataset/features_having_most_influence_on_Botnet_IoT.csv")
```

```
# data=pd.read_csv(r"/content/drive/MyDrive/iot_botnet_dataset/features_having_most_influence_on_Botnet_IoT.csv")
data=pd.read_csv(r"/content/drive/MyDrive/Group 10 - IBM Project - Sandeep and Sudhay/Temp_Folders/Datasets/features_having_most_influence_on_E
```

```
data.head()
```

	pkSeqID	proto	saddr	sport	daddr	dport	seq	stddev	N_IN_Conn_P_SrcIP	min	state_number	mean	N_IN_(
0	792371	udp	192.168.100.150	48516	192.168.100.3	80	175094	0.226784	100	4.100436	4	4.457383	
1	2056418	tcp	192.168.100.148	22267	192.168.100.3	80	143024	0.451998	100	3.439257	1	3.806172	

```
# encoder1 = ce.HashingEncoder(cols='saddr',n_components=5)
# data = encoder1.fit_transform(data)
# encoder2 = ce.HashingEncoder(cols='daddr',n_components=5)
# data = encoder2.fit_transform(data)
```

```
len(data)
```

```
733705
```

```
data.dtypes
```

```
pkSeqID                int64
proto                 object
saddr                 object
sport                 object
daddr                 object
dport                 object
seq                    int64
stddev               float64
N_IN_Conn_P_SrcIP      int64
min                  float64
state_number           int64
mean                 float64
N_IN_Conn_P_DstIP      int64
drate                float64
srate                float64
max                  float64
attack                 int64
category              object
subcategory           object
dtype: object
```

```
df = data.drop(['pkSeqID','subcategory','attack','dport','sport'],axis=1)
```

```
#encoder1 = ce.HashingEncoder(cols='saddr',n_components=6)

encoder1 = ce.BinaryEncoder(cols=['saddr'],return_df=True)
df = encoder1.fit_transform(df)
```

```
encoder2 = ce.BinaryEncoder(cols=['daddr'],return_df=True)
df = encoder2.fit_transform(df)
```

```
proto_encoded = pd.get_dummies(data=df['proto'],drop_first=True)
```

```
df = pd.concat([df,proto_encoded],axis=1)
df.drop('proto',axis=1,inplace=True)
```

```
df.dtypes
```

```
saddr_0                int64
saddr_1                int64
saddr_2                int64
saddr_3                int64
saddr_4                int64
daddr_0                int64
daddr_1                int64
daddr_2                int64
daddr_3                int64
daddr_4                int64
daddr_5                int64
seq                    int64
stddev               float64
N_IN_Conn_P_SrcIP      int64
min                  float64
state_number           int64
mean                 float64
N_IN_Conn_P_DstIP      int64
drate                float64
srate                float64
max                  float64
category              object
icmp                   uint8
ipv6-icmp              uint8
tcp                    uint8
udp                    uint8
dtype: object
```

```
scaler = StandardScaler()
X = df.drop('category',axis=1)
y = df['category']
X = scaler.fit_transform(X)
```

```
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=0.8,random_state=109)
```

```
clf = LogisticRegression(random_state=0,multi_class='multinomial').fit(X_train, y_train)
```

```
    /usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: lbfgs failed to converge (status=1):
    STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

    Increase the number of iterations (max_iter) or scale the data as shown in:
        https://scikit-learn.org/stable/modules/preprocessing.html
    Please also refer to the documentation for alternative solver options:
        https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
      extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
predictions = clf.predict(X_test)
print(accuracy_score(y_test,predictions))
```

```
    0.9853347053652354
```

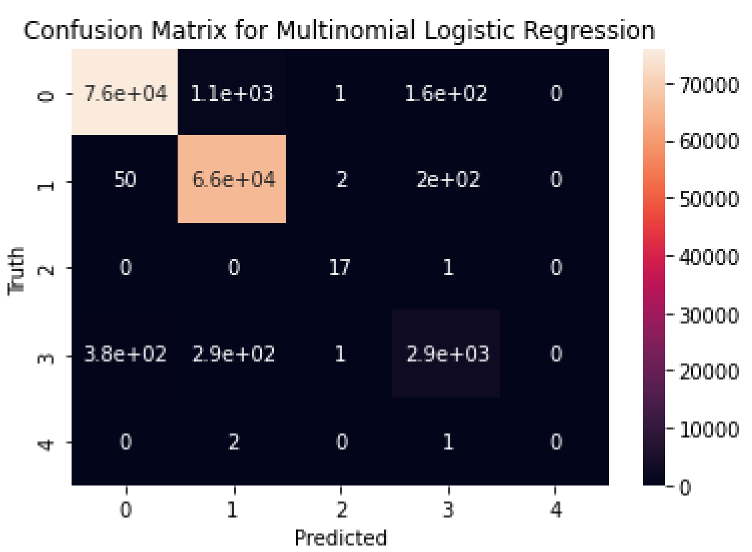
```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
print("\t\tClassification report for Multinomial Logistic Regression\n\n",classification_report(y_test,predictions,digits=6))
print()
print()
plt.title("Confusion Matrix for Multinomial Logistic Regression")
sns.heatmap(confusion_matrix(y_test,predictions),annot=True)
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```

```
    /usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision and F-score are ill-def
      _warn_prf(average, modifier, msg_start, len(result))
    /usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision and F-score are ill-def
      _warn_prf(average, modifier, msg_start, len(result))
    /usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308: UndefinedMetricWarning: Precision and F-score are ill-def
      _warn_prf(average, modifier, msg_start, len(result))
                    Classification report for Multinomial Logistic Regression

                     precision    recall  f1-score   support

               DDoS   0.994369  0.984061  0.989188     76983
                DoS   0.979881  0.996145  0.987946     66152
             Normal   0.809524  0.944444  0.871795        18
     Reconnaissance   0.888584  0.814226  0.849782      3585
              Theft   0.000000  0.000000  0.000000         3

           accuracy                       0.985335    146741
          macro avg   0.734472  0.747775  0.739742    146741
       weighted avg   0.985210  0.985335  0.985188    146741
```




```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

```
from sklearn.preprocessing import OneHotEncoder
ohe = OneHotEncoder()
y = df['category'].values
```

```
y = ohe.fit_transform(y.reshape(-1,1)).toarray()
X_train,X_test,y_train,y_test = train_test_split(X,y,train_size=0.8,random_state=109)
```

```
model = Sequential()
model.add(Dense(16,input_dim=25,activation='relu'))
model.add(Dense(12,activation='relu'))
model.add(Dense(5,activation='softmax'))
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
history = model.fit(X_train, y_train, epochs=100, batch_size=64)
```

```
Epoch 1/100
9172/9172 [==============================] - 18s 2ms/step - loss: 0.0626 - accuracy: 0.9798
Epoch 2/100
9172/9172 [==============================] - 19s 2ms/step - loss: 0.0254 - accuracy: 0.9901
Epoch 3/100
9172/9172 [==============================] - 18s 2ms/step - loss: 0.0233 - accuracy: 0.9907
Epoch 4/100
9172/9172 [==============================] - 17s 2ms/step - loss: 0.0219 - accuracy: 0.9914
Epoch 5/100
9172/9172 [==============================] - 16s 2ms/step - loss: 0.0206 - accuracy: 0.9919
Epoch 6/100
9172/9172 [==============================] - 17s 2ms/step - loss: 0.0194 - accuracy: 0.9925
Epoch 7/100
9172/9172 [==============================] - 17s 2ms/step - loss: 0.0185 - accuracy: 0.9928
Epoch 8/100
9172/9172 [==============================] - 18s 2ms/step - loss: 0.0175 - accuracy: 0.9932
Epoch 9/100
9172/9172 [==============================] - 15s 2ms/step - loss: 0.0165 - accuracy: 0.9935
Epoch 10/100
9172/9172 [==============================] - 16s 2ms/step - loss: 0.0157 - accuracy: 0.9940
Epoch 11/100
9172/9172 [==============================] - 16s 2ms/step - loss: 0.0149 - accuracy: 0.9942
Epoch 12/100
9172/9172 [==============================] - 17s 2ms/step - loss: 0.0143 - accuracy: 0.9945
Epoch 13/100
9172/9172 [==============================] - 19s 2ms/step - loss: 0.0141 - accuracy: 0.9950
Epoch 14/100
9172/9172 [==============================] - 19s 2ms/step - loss: 0.0127 - accuracy: 0.9951
Epoch 15/100
9172/9172 [==============================] - 16s 2ms/step - loss: 0.0122 - accuracy: 0.9953
Epoch 16/100
9172/9172 [==============================] - 17s 2ms/step - loss: 0.0119 - accuracy: 0.9954
Epoch 17/100
9172/9172 [==============================] - 17s 2ms/step - loss: 0.0113 - accuracy: 0.9955
Epoch 18/100
9172/9172 [==============================] - 17s 2ms/step - loss: 0.0116 - accuracy: 0.9957
Epoch 19/100
9172/9172 [==============================] - 16s 2ms/step - loss: 0.0109 - accuracy: 0.9958
Epoch 20/100
9172/9172 [==============================] - 16s 2ms/step - loss: 0.0105 - accuracy: 0.9959
Epoch 21/100
9172/9172 [==============================] - 15s 2ms/step - loss: 0.0110 - accuracy: 0.9959
Epoch 22/100
9172/9172 [==============================] - 16s 2ms/step - loss: 0.0103 - accuracy: 0.9959
Epoch 23/100
9172/9172 [==============================] - 17s 2ms/step - loss: 0.0102 - accuracy: 0.9960
Epoch 24/100
9172/9172 [==============================] - 17s 2ms/step - loss: 0.0098 - accuracy: 0.9961
Epoch 25/100
9172/9172 [==============================] - 15s 2ms/step - loss: 0.0097 - accuracy: 0.9963
Epoch 26/100
9172/9172 [==============================] - 16s 2ms/step - loss: 0.0094 - accuracy: 0.9962
Epoch 27/100
9172/9172 [==============================] - 15s 2ms/step - loss: 0.0093 - accuracy: 0.9962
Epoch 28/100
9172/9172 [==============================] - 16s 2ms/step - loss: 0.0114 - accuracy: 0.9963
Epoch 29/100
9172/9172 [==============================] - 15s 2ms/step - loss: 0.0094 - accuracy: 0.9963
```

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

```
pred = list()
for i in range(len(y_pred)):
    pred.append(np.argmax(y_pred[i]))
test = list()
for i in range(len(y_test)):
    test.append(np.argmax(y_test[i]))
```

```
from sklearn.metrics import accuracy_score
a = accuracy_score(test,pred)
print('Accuracy is:', a)
```

```
Accuracy is: 0.9976966219393353
```

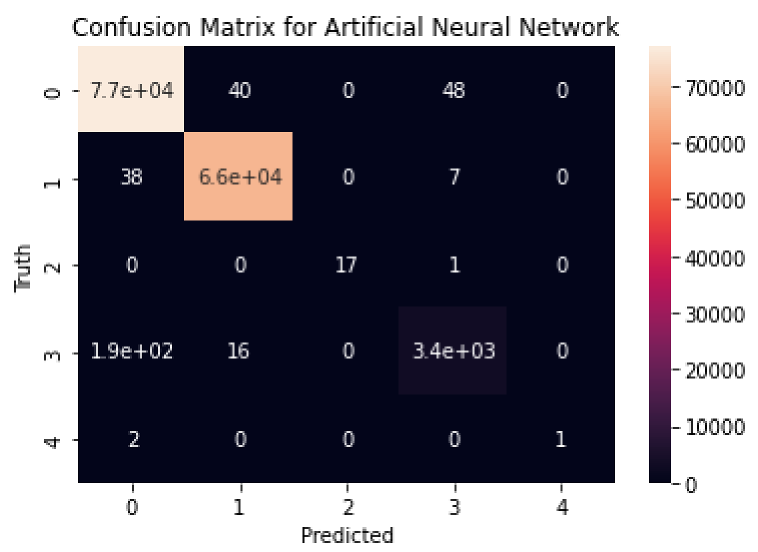
```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
print("\t\tClassification report for Artificial Neural Network\n\n",classification_report(test,pred,digits=6))
print()
print()
plt.title("Confusion Matrix for Artificial Neural Network")
sns.heatmap(confusion_matrix(test,pred),annot=True)
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```

```
                    Classification report for Artificial Neural Network

                   precision    recall  f1-score   support

               0   0.997070  0.998857  0.997962     76983
               1   0.999154  0.999320  0.999237     66152
               2   1.000000  0.944444  0.971429        18
               3   0.983716  0.943654  0.963269      3585
               4   1.000000  0.333333  0.500000         3

        accuracy                       0.997697    146741
       macro avg   0.995988  0.843922  0.886379    146741
    weighted avg   0.997683  0.997697  0.997676    146741
```



Support Vector Machine - SVClassifier

```
from sklearn.svm import SVC
model = SVC(kernel="linear")
model.fit(X_train,y_train)
```

```
print('Accuracy is:', model.score(X_test,y_test)*100)
```

GridSearch with SVC

```
from sklearn.model_selection import GridSearchCV
param_grid = {'C':(1, 10, 100, 1000), 'gamma':(0.1, 0.01, 0.001, 0.0001), 'kernel':('rbf','linear','poly')}
grid_search = GridSearchCV(model, param_grid, cv=10, verbose=10)
grid_search.fit(X_train, y_train)
```

```
grid_search.best_params_
```

Decision Tree Classifier with GridSearch

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
dec = DecisionTreeClassifier()
params = {'max_depth':(1,2,3,10,100,1000)}
grid_search_dec = GridSearchCV(dec, params, cv=5, verbose=10, n_jobs=-1)
grid_search_dec.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 6 candidates, totalling 30 fits
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=-1,
```

```
                    param_grid={'max_depth': (1, 2, 3, 10, 100, 1000)}, verbose=10)
```

```
grid_search_dec.best_params_
```

```
{'max_depth': 1000}
```

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

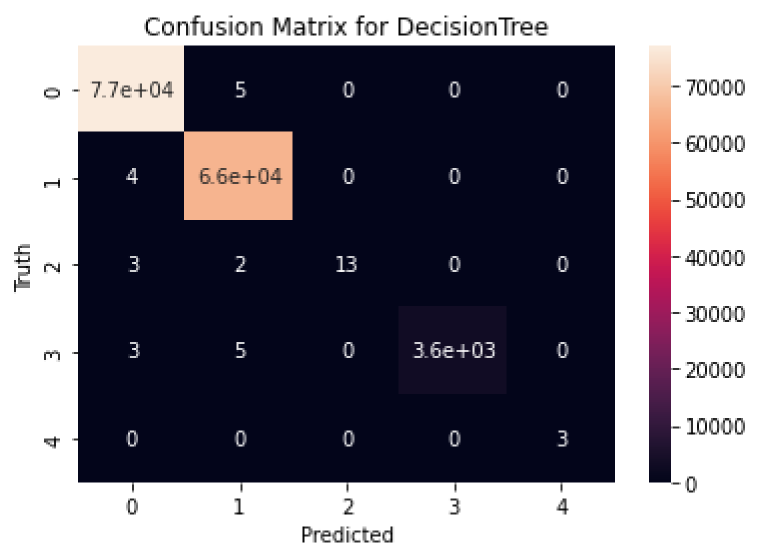
```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
print("\t\tClassification report for DecisionTree\n\n",classification_report(y_test,y_pred,digits=6))
print()
print()
plt.title("Confusion Matrix for DecisionTree")
sns.heatmap(confusion_matrix(y_test,y_pred),annot=True)
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```

```
                 Classification report for DecisionTree

                  precision    recall  f1-score   support

          DDoS   0.999870  0.999935  0.999903     76983
           DoS   0.999819  0.999940  0.999879     66152
        Normal   1.000000  0.722222  0.838710        18
Reconnaissance   1.000000  0.997768  0.998883      3585
         Theft   1.000000  1.000000  1.000000         3

      accuracy                       0.999850    146741
     macro avg   0.999938  0.943973  0.967475    146741
  weighted avg   0.999850  0.999850  0.999847    146741
```



Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
randForest = RandomForestClassifier()
params = {'max_depth':(1,2,9,10),'n_estimators':(10,15,30)}
grid_search = GridSearchCV(randForest, params, cv=5, verbose=10)
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[CV 1/5; 1/12] START max_depth=1, n_estimators=10...............................
[CV 1/5; 1/12] END max_depth=1, n_estimators=10;, score=0.854 total time=   3.8s
[CV 2/5; 1/12] START max_depth=1, n_estimators=10...............................
[CV 2/5; 1/12] END max_depth=1, n_estimators=10;, score=0.796 total time=   3.9s
[CV 3/5; 1/12] START max_depth=1, n_estimators=10...............................
[CV 3/5; 1/12] END max_depth=1, n_estimators=10;, score=0.832 total time=   4.3s
[CV 4/5; 1/12] START max_depth=1, n_estimators=10...............................
[CV 4/5; 1/12] END max_depth=1, n_estimators=10;, score=0.751 total time=   8.4s
[CV 5/5; 1/12] START max_depth=1, n_estimators=10...............................
[CV 5/5; 1/12] END max_depth=1, n_estimators=10;, score=0.852 total time=   7.3s
[CV 1/5; 2/12] START max_depth=1, n_estimators=15...............................
[CV 1/5; 2/12] END max_depth=1, n_estimators=15;, score=0.857 total time=   8.4s
[CV 2/5; 2/12] START max_depth=1, n_estimators=15...............................
[CV 2/5; 2/12] END max_depth=1, n_estimators=15;, score=0.852 total time=   8.2s
[CV 3/5; 2/12] START max_depth=1, n_estimators=15...............................
[CV 3/5; 2/12] END max_depth=1, n_estimators=15;, score=0.861 total time=   7.4s
[CV 4/5; 2/12] START max_depth=1, n_estimators=15...............................
[CV 4/5; 2/12] END max_depth=1, n_estimators=15;, score=0.840 total time=   6.7s
[CV 5/5; 2/12] START max_depth=1, n_estimators=15...............................
[CV 5/5; 2/12] END max_depth=1, n_estimators=15;, score=0.817 total time=   5.1s
[CV 1/5; 3/12] START max_depth=1, n_estimators=30...............................
```

```
[CV 1/5; 3/12] END max_depth=1, n_estimators=30;, score=0.838 total time=   8.5s
[CV 2/5; 3/12] START max_depth=1, n_estimators=30..............................
[CV 2/5; 3/12] END max_depth=1, n_estimators=30;, score=0.862 total time=   8.8s
[CV 3/5; 3/12] START max_depth=1, n_estimators=30..............................
[CV 3/5; 3/12] END max_depth=1, n_estimators=30;, score=0.831 total time=   8.5s
[CV 4/5; 3/12] START max_depth=1, n_estimators=30..............................
[CV 4/5; 3/12] END max_depth=1, n_estimators=30;, score=0.863 total time=   8.2s
[CV 5/5; 3/12] START max_depth=1, n_estimators=30..............................
[CV 5/5; 3/12] END max_depth=1, n_estimators=30;, score=0.841 total time=   8.3s
[CV 1/5; 4/12] START max_depth=2, n_estimators=10..............................
[CV 1/5; 4/12] END max_depth=2, n_estimators=10;, score=0.903 total time=   5.5s
[CV 2/5; 4/12] START max_depth=2, n_estimators=10..............................
[CV 2/5; 4/12] END max_depth=2, n_estimators=10;, score=0.849 total time=   5.4s
[CV 3/5; 4/12] START max_depth=2, n_estimators=10..............................
[CV 3/5; 4/12] END max_depth=2, n_estimators=10;, score=0.904 total time=   5.1s
[CV 4/5; 4/12] START max_depth=2, n_estimators=10..............................
[CV 4/5; 4/12] END max_depth=2, n_estimators=10;, score=0.876 total time=   5.1s
[CV 5/5; 4/12] START max_depth=2, n_estimators=10..............................
[CV 5/5; 4/12] END max_depth=2, n_estimators=10;, score=0.844 total time=   6.2s
[CV 1/5; 5/12] START max_depth=2, n_estimators=15..............................
[CV 1/5; 5/12] END max_depth=2, n_estimators=15;, score=0.911 total time=   7.5s
[CV 2/5; 5/12] START max_depth=2, n_estimators=15..............................
[CV 2/5; 5/12] END max_depth=2, n_estimators=15;, score=0.880 total time=   7.3s
[CV 3/5; 5/12] START max_depth=2, n_estimators=15..............................
[CV 3/5; 5/12] END max_depth=2, n_estimators=15;, score=0.912 total time=   6.6s
[CV 4/5; 5/12] START max_depth=2, n_estimators=15..............................
[CV 4/5; 5/12] END max_depth=2, n_estimators=15;, score=0.911 total time=   6.9s
[CV 5/5; 5/12] START max_depth=2, n_estimators=15..............................
[CV 5/5; 5/12] END max_depth=2, n_estimators=15;, score=0.862 total time=   7.3s
[CV 1/5; 6/12] START max_depth=2, n_estimators=30..............................
[CV 1/5; 6/12] END max_depth=2, n_estimators=30;, score=0.903 total time=  13.5s
[CV 2/5; 6/12] START max_depth=2, n_estimators=30..............................
[CV 2/5; 6/12] END max_depth=2, n_estimators=30;, score=0.903 total time=  13.4s
[CV 3/5; 6/12] START max_depth=2, n_estimators=30..............................
[CV 3/5; 6/12] END max_depth=2, n_estimators=30;, score=0.893 total time=  13.0s
[CV 4/5; 6/12] START max_depth=2, n_estimators=30..............................
```

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

```
grid_search.best_params_
```

```
{'max_depth': 10, 'n_estimators': 10}
```

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
print("\t\tClassification report for RandomForest\n\n",classification_report(y_test,y_pred,digits=6))
print()
print()
plt.title("Confusion Matrix for RandomForest")
sns.heatmap(confusion_matrix(y_test,y_pred),annot=True)
plt.xlabel("Predicted")
plt.ylabel("Truth")
plt.show()
```

```
                 Classification report for RandomForest

                  precision    recall  f1-score   support

           DDoS   0.999545  0.999662  0.999604     76983
            DoS   0.999562  0.999516  0.999539     66152
         Normal   1.000000  0.722222  0.838710        18
 Reconnaissance   0.999163  0.998884  0.999024      3585
          Theft   1.000000  1.000000  1.000000         3

       accuracy                       0.999543    146741
      macro avg   0.999654  0.944057  0.967375    146741
   weighted avg   0.999543  0.999543  0.999541    146741
```