ADFA-LD - Model Evaluation

In [1]:

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from IPython.display import display
pd.options.display.max_columns = None
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier
from IPython.display import display
from sklearn import metrics
from sklearn.model_selection import train_test_split
import statistics
import numpy as np
from sklearn import metrics
from sklearn.preprocessing import MinMaxScaler, StandardScaler, LabelEncoder
from sklearn.feature_selection import SelectKBest
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
```

In [2]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.callbacks import EarlyStopping
from keras.layers import Dropout
import re
```

```
Using TensorFlow backend.
```

In [3]:

```
import glob
import math
from collections import Counter
import csv

import numpy as np


def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):
    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / float(np.sum(cm))
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy, m
isclass))
    plt.show()

# returns a dictionary of n-grams frequency for any list
def ngrams_freq(listname, n):
    counts = dict()
    # make n-grams as string iteratively
    grams = [' '.join(listname[i:i+n]) for i in range(len(listname)-n)]
    for gram in grams:
```

```
        if gram not in counts:
            counts[gram] = 1
        else:
            counts[gram] += 1
    return counts

# returns the values of features for any list
def feature_freq(listname,n,features):
        counts = dict()
        # make n-grams as string iteratively
        grams = [' '.join(listname[i:i+n]) for i in range(len(listname)-n)]
        for gram in grams:
                counts[gram] = 0
        for gram in grams:
                if gram in features:
                        counts[gram] += 1
        return counts

# values of n for finding n-grams
n_values = [1]

# Base address for attack data files
add = "ADFA-LD/ADFA-LD/Attack_Data_Master/"
# list of attacks
attack = ['Adduser','Hydra_FTP','Hydra_SSH','Java_Meterpreter','Meterpreter','Web_Shel
l']

# initializing dictionary for n-grams from all files
traindict = {}

Attack_list_new = []
print("Generating Training Data ..................................")
for term in attack:
        print(" Training data from " + term)
        globals()['%s_list' % term] = []
        in_address = add+term
        k = 1
        # finding list of data from all files
        for i in range (1,11):
                read_files = glob.glob(in_address+"_"+str(i)+"/*.txt")
                for f in read_files:
                        with open(f, "r") as infile:
                                globals()['%s_list_array' % term+str(k)] = ALine =infil
e.read()
                                #ALine = ALine[:820]
                                Attack_list_new.append(term +','+ str(ALine))
                                globals()['%s_list' % term].extend(globals()['%s_list_a
rray' % term+str(k)])
                                k += 1
        # number of lists for distinct files
        globals()['%s_size' % term] = k-1
        # combined list of all files
        listname = globals()['%s_list' % term]
        # finding n-grams and extracting top 30%
        for n in n_values:
                #print("                Extracting top 30% "+str(n)+"-grams from "+term
+"......................")
                dictname = ngrams_freq(listname,n)
                top = math.ceil(0.3*len(dictname))
                dictname = Counter(dictname)
                for k, v in dictname.most_common(top):
```

```
                        traindict.update({k : v})

# finding training data for Normal file
print(" Training data from Normal")
Normal_list = []
Normal_list_new = []
in_address = "ADFA-LD/ADFA-LD/Training_Data_Master/"
k = 1
read_files = glob.glob(in_address+"/*.txt")
for f in read_files:
        with open(f, "r") as infile:
                globals()['Normal%s_list_array' % str(k)] = Line = infile.read()
                Normal_list_new.append('Normal,'+ str(Line))
                Normal_list.extend(globals()['Normal%s_list_array' % str(k)])
                k += 1

# number of lists for distinct files
Normal_list_size = k-1
# combined list of all files
listname = Normal_list


print("\nnew_train.csv created..............................................\n")
```

```
Generating Training Data .................................
        Training data from Adduser
        Training data from Hydra_FTP
        Training data from Hydra_SSH
        Training data from Java_Meterpreter
        Training data from Meterpreter
        Training data from Web_Shell
        Training data from Normal

new_train.csv created..............................................
```

In [4]:

```
new_train_list = []
new_train_list = Normal_list_new + Attack_list_new
#new_train_list[1]
#Attack_list_new[1]
```

In [5]:

```
new_train_list = []
new_train_list = Normal_list_new + Attack_list_new


with open('new_train.csv', 'w') as f:
    for item in new_train_list:
        f.write("%s\n" % item)
```

In [6]:

```
train = pd.read_csv("./new_train.csv", sep=',',error_bad_lines=False, header=None, name
s=['Label','CallTrace'])
train.head(5)
train.shape
#train.info()

#train.describe(include = 'all')
train_df = train.copy()
train['Label'] = train['Label'].astype('category')
train['CallTrace'] = train['CallTrace'].astype('category')

train['Label'].value_counts()
#train['CallTrace'].value_counts()
```

Out[6]:

```
Normal              833
Hydra_SSH           176
Hydra_FTP           162
Java_Meterpreter    124
Web_Shell           118
Adduser              91
Meterpreter          75
Name: Label, dtype: int64
```

In [7]:

```
train['Label_Codes'] = train['Label'].cat.codes
train['CallTrace_Codes'] = train['CallTrace'].cat.codes
train['Label_Codes'].value_counts()
```

Out[7]:

```
5    833
2    176
1    162
3    124
6    118
0     91
4     75
Name: Label_Codes, dtype: int64
```

In [8]:

```
train.head()
```

Out[8]:

	Label	CallTrace	Label_Codes	CallTrace_Codes
0	Normal	6 6 63 6 42 120 6 195 120 6 6 114 114 1 1 252 ...	5	1407
1	Normal	54 175 120 175 175 3 175 175 120 175 120 175 1...	5	1239
2	Normal	6 11 45 33 192 33 5 197 192 6 33 5 3 197 192 1...	5	1286
3	Normal	7 174 174 5 197 197 6 13 195 4 4 118 6 91 38 5...	5	1465
4	Normal	11 45 33 192 33 5 197 192 6 33 5 3 197 192 192...	5	93

Multinominal Logistic Regression

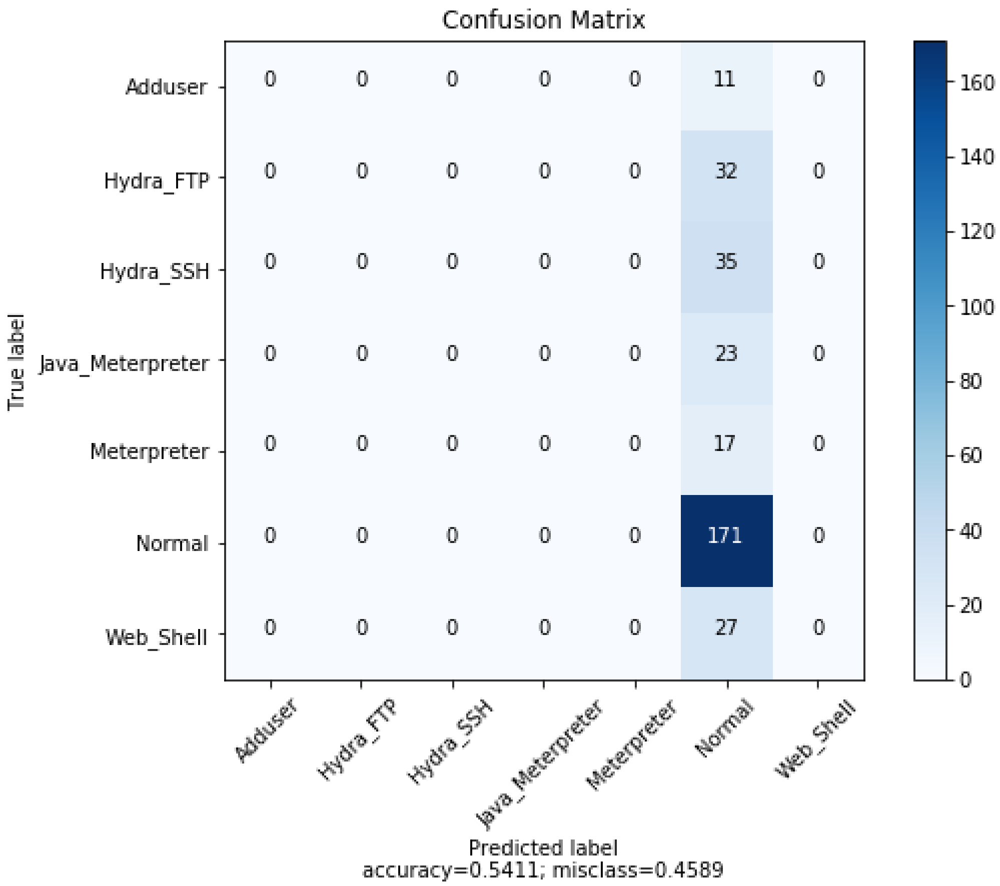
In [9]:

```
import warnings
warnings.filterwarnings("ignore")

# split the dataset in train and test
X = train.iloc[:, [3]].values
y = train.iloc[:, 2].values


# Splitting the dataset into the Training set and Test set
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
= 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(multi_class='ovr', solver = 'lbfgs')
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# How did our model perform?
from sklearn import metrics
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
```

```
Misclassified samples: 145
Accuracy: 0.54
```

In [10]:

```
#classifier.predict_proba(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cm,
                      normalize    = False,
                      target_names = ['Adduser', 'Hydra_FTP', 'Hydra_SSH', 'Java_Meterp
reter', 'Meterpreter', 'Normal', 'Web_Shell'],
                      title        = "Confusion Matrix")
```



Logistic Regression Binary Classification

In [11]:

```
train.loc[train.Label != 'Normal','Label_Binary']= 1
train.loc[train.Label == 'Normal','Label_Binary']= 0
train['Label_Binary'].value_counts()
#train.head()
```

Out[11]:

```
0.0    833
1.0    746
Name: Label_Binary, dtype: int64
```

In [12]:

```
train.head()
```

Out[12]:

	Label	CallTrace	Label_Codes	CallTrace_Codes	Label_Binary
0	Normal	6 6 63 6 42 120 6 195 120 6 6 114 114 1 1 252 ...	5	1407	0.0
1	Normal	54 175 120 175 175 3 175 175 120 175 120 175 1...	5	1239	0.0
2	Normal	6 11 45 33 192 33 5 197 192 6 33 5 3 197 192 1...	5	1286	0.0
3	Normal	7 174 174 5 197 197 6 13 195 4 4 118 6 91 38 5...	5	1465	0.0
4	Normal	11 45 33 192 33 5 197 192 6 33 5 3 197 192 192...	5	93	0.0

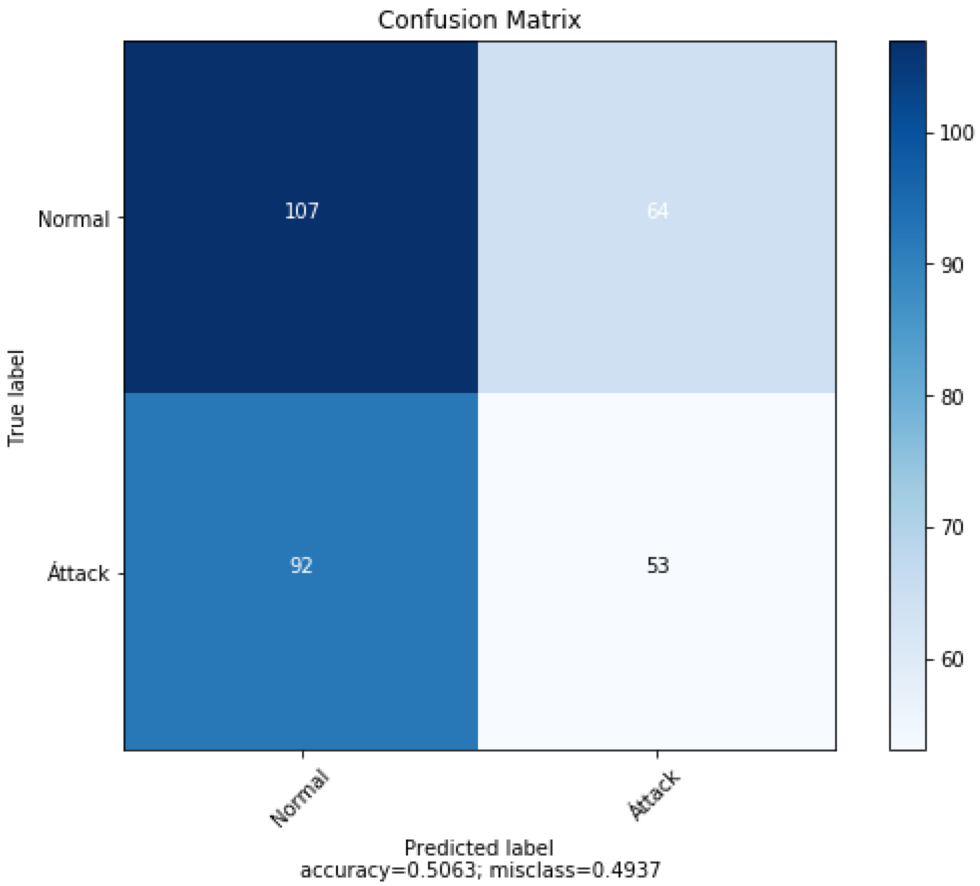
In [13]:

```
import warnings
warnings.filterwarnings("ignore")

# split the dataset in train and test
X = train.iloc[:, [3]].values
y = train.iloc[:, 4].values


# Splitting the dataset into the Training set and Test set
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
= 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
#classifier = LogisticRegression(multi_class='ovr', solver = 'lbfgs')
classifier = LogisticRegression()
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# How did our model perform?
from sklearn import metrics
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
```

```
Misclassified samples: 156
Accuracy: 0.51
```

In [14]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cm,
                      normalize    = False,
                      target_names = ['Normal', 'Áttack'],
                      title        = "Confusion Matrix")
```



In [15]:

```
print(metrics.classification_report(y_pred, y_test))
```

```
              precision    recall  f1-score   support

         0.0       0.63      0.54      0.58       199
         1.0       0.37      0.45      0.40       117

   micro avg       0.51      0.51      0.51       316
   macro avg       0.50      0.50      0.49       316
weighted avg       0.53      0.51      0.51       316
```

OneHotEncoding for LogisticRegression

In [16]:

```
# Split into predictor and response dataframes.
train_df_enc = train_df.copy()
X_df = train_df_enc.drop('Label', axis=1)
y = train_df_enc['Label']

X_df.shape,y.shape
```

Out[16]:

```
((1579, 1), (1579,))
```

In [17]:

```
X_df.head()
```

Out[17]:

	CallTrace
0	6 6 63 6 42 120 6 195 120 6 6 114 114 1 1 252 ...
1	54 175 120 175 175 3 175 175 120 175 120 175 1...
2	6 11 45 33 192 33 5 197 192 6 33 5 3 197 192 1...
3	7 174 174 5 197 197 6 13 195 4 4 118 6 91 38 5...
4	11 45 33 192 33 5 197 192 6 33 5 3 197 192 192...

In [18]:

```
train_df.head()
```

Out[18]:

	Label	CallTrace
0	Normal	6 6 63 6 42 120 6 195 120 6 6 114 114 1 1 252 ...
1	Normal	54 175 120 175 175 3 175 175 120 175 120 175 1...
2	Normal	6 11 45 33 192 33 5 197 192 6 33 5 3 197 192 1...
3	Normal	7 174 174 5 197 197 6 13 195 4 4 118 6 91 38 5...
4	Normal	11 45 33 192 33 5 197 192 6 33 5 3 197 192 192...

In [19]:

```
# Map response variable to integers 0,1.
y = pd.Series(np.where(y.values != 'Normal',1,0), y.index)
y.value_counts()
```

Out[19]:

```
0    833
1    746
dtype: int64
```

In [20]:

```
# Label Encode instead of dummy variables

mappings = []

from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

label_df = train.drop('Label', axis=1)
label_df = train.drop('Label_Binary', axis=1)
label_df = train.drop('Label_Codes', axis=1)
label_df['CallTrace'] = label_df['CallTrace_Codes']
label_df = X_df.copy()
for i, col in enumerate(label_df):
    if label_df[col].dtype == 'object':
        label_df[col] = label_encoder.fit_transform(np.array(label_df[col].astype(str))
.reshape((-1,)))
        mappings.append(dict(zip(label_encoder.classes_, range(1, len(label_encoder.cla
sses_)+1))))
```

In [21]:

```
label_df.head()
```

Out[21]:

	CallTrace
0	1407
1	1239
2	1286
3	1465
4	93

In [22]:

```
from sklearn.preprocessing import OneHotEncoder


onehot_encoder = OneHotEncoder()
for i, col in enumerate(label_df):
    if label_df[col].dtype == 'object':
        label_df[col] = onehot_encoder.fit_transform(np.array(label_df[col].astype(str
)).reshape((-1,)))
        mappings.append(dict(zip(onehot_encoder.classes_, range(1, len(onehot_encoder.c
lasses_)+1))))
```

In [23]:

```
 label_df[col].head()
```

Out[23]:

```
0    1407
1    1239
2    1286
3    1465
4      93
Name: CallTrace, dtype: int32
```

In [24]:

```
X_train, X_test, y_train, y_test = train_test_split(label_df, y, test_size = 0.2, rando
m_state = 10)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[24]:

```
((1263, 1), (316, 1), (1263,), (316,))
```

In [25]:

```
clf = LogisticRegression()
model_mix = clf.fit(X_train, y_train)
# y_pred = model_norm.predict(X_test)
print("Model accuracy is", model_mix.score(X_test, y_test))
```

```
Model accuracy is 0.5569620253164557
```

In [26]:

```
model_mix
```

Out[26]:

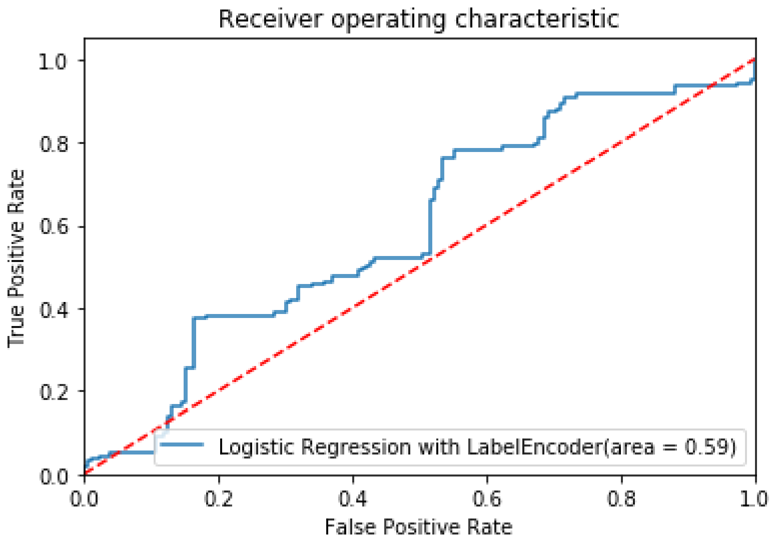
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

In [27]:

```
# logit_roc_auc = roc_auc_score(y_test, model_norm.predict(X_test))
# fpr, tpr, thresholds = roc_curve(y_test, model_norm.predict_proba(X_test)[:,1])

classes = model_mix.predict(X_test)
probs = model_mix.predict_proba(X_test)
preds = probs[:,1]
#preds
```

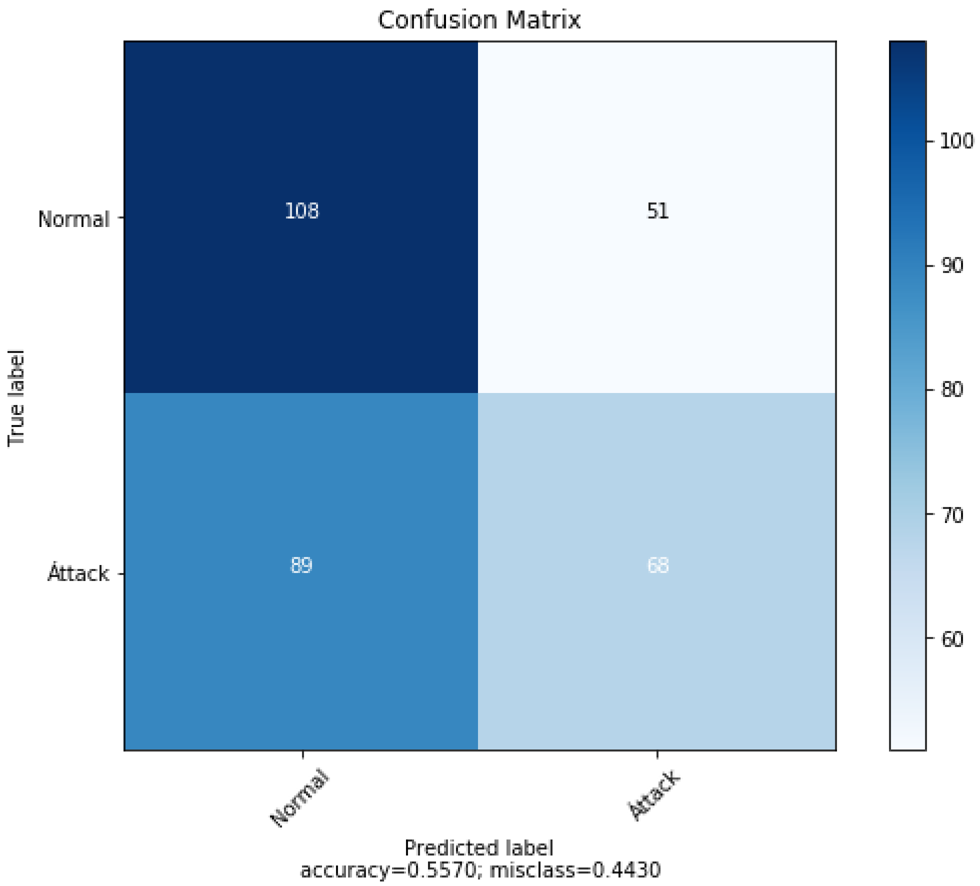
In [28]:

```
labelfpr, labeltpr, labelthreshold = metrics.roc_curve(y_test, preds)
label_roc_auc = metrics.auc(labelfpr, labeltpr)

plt.figure()
plt.plot(labelfpr, labeltpr, label='Logistic Regression with LabelEncoder(area = %0.2f
)' % label_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



In [29]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, classes)
plot_confusion_matrix(cm,
                      normalize    = False,
                      target_names = ['Normal', 'Áttack'],
                      title        = "Confusion Matrix")
```



In [30]:

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[30]:

```
((1263, 1), (316, 1), (1263,), (316,))
```

In [31]:

```
print(metrics.classification_report(classes, y_test))
```

```
              precision    recall  f1-score   support

           0       0.68      0.55      0.61       197
           1       0.43      0.57      0.49       119

   micro avg       0.56      0.56      0.56       316
   macro avg       0.56      0.56      0.55       316
weighted avg       0.59      0.56      0.56       316
```

RandomForest Classification

In [32]:

```
# Normalize using MinMaxScaler to constrain values to between 0 and 1.
from sklearn.preprocessing import MinMaxScaler, StandardScaler

scaler = MinMaxScaler(feature_range = (0,1))

scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

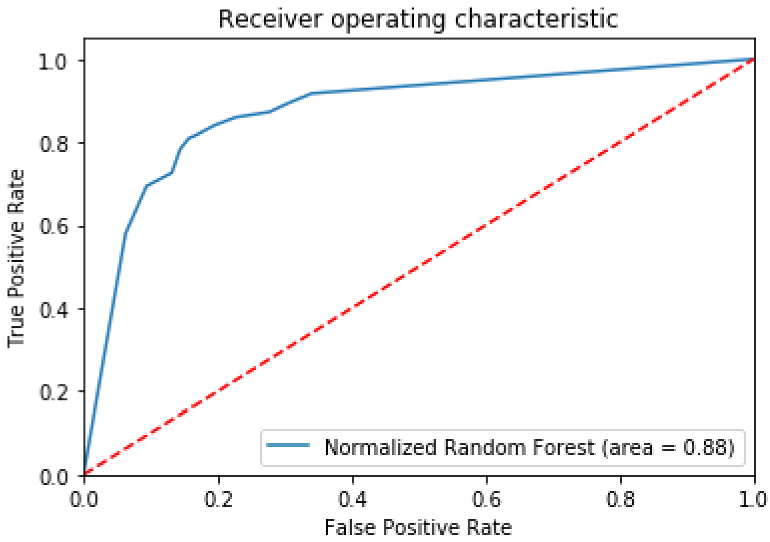
In [33]:

```
clf = RandomForestClassifier(n_jobs=-1)
model_rf = clf.fit(X_train, y_train)
print('Model accuracy is',model_rf.score(X_test, y_test))
```

```
Model accuracy is 0.8259493670886076
```

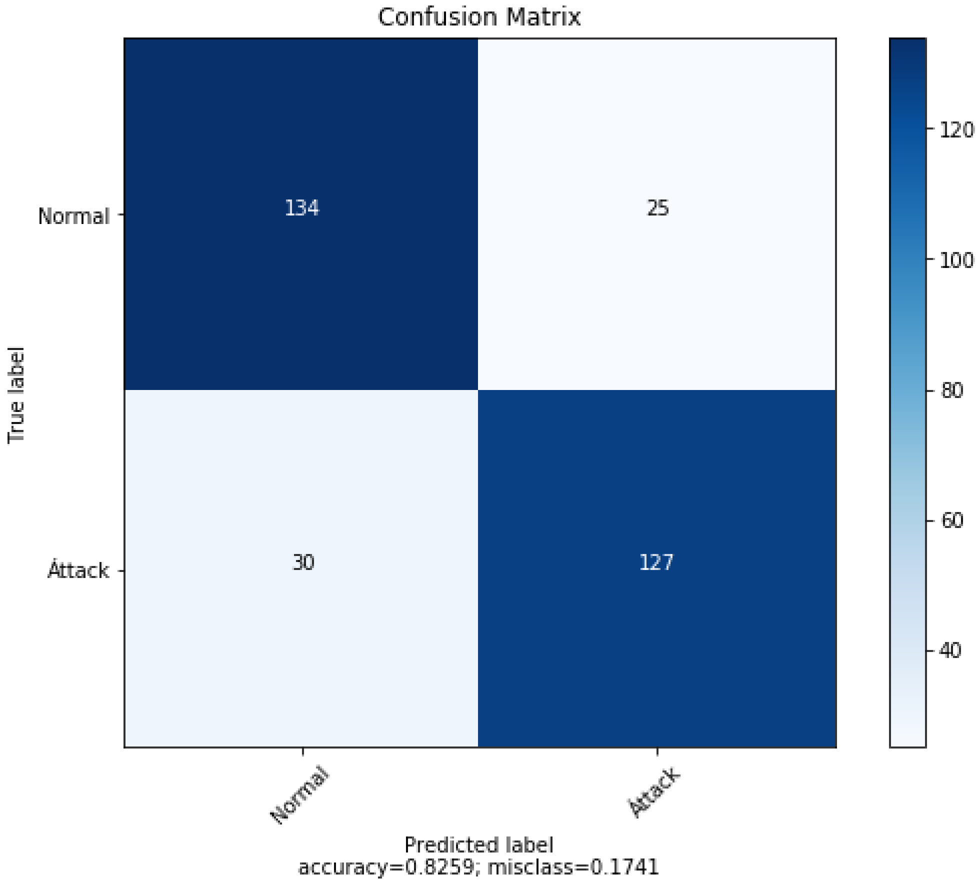
In [34]:

```
probs = model_rf.predict_proba(X_test)
preds = probs[:,1]
rffpr, rftpr, rfthreshold = metrics.roc_curve(y_test, preds)
rf_roc_auc = metrics.auc(rffpr, rftpr)

plt.figure()
plt.plot(rffpr, rftpr, label='Normalized Random Forest (area = %0.2f)' % rf_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```



In [35]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
classes = model_rf.predict(X_test)
cm = confusion_matrix(y_test, classes)
plot_confusion_matrix(cm,
                      normalize    = False,
                      target_names = ['Normal', 'Áttack'],
                      title        = "Confusion Matrix")
```



In [36]:

```
print(metrics.classification_report(classes, y_test))
```

```
              precision    recall  f1-score   support

           0       0.84      0.82      0.83       164
           1       0.81      0.84      0.82       152

   micro avg       0.83      0.83      0.83       316
   macro avg       0.83      0.83      0.83       316
weighted avg       0.83      0.83      0.83       316
```

Train Data with ngrams

In [37]:

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix, recall_score, roc_auc_sco
re, precision_score

X, y = make_classification(
    n_classes=2, class_sep=1.5, weights=[0.1, 0.9],
    n_features=20, n_samples=1000, random_state=10
)

#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state
=42)

clf = LogisticRegression(class_weight="balanced")
clf.fit(X_train, y_train)
THRESHOLD = 0.5
preds = np.where(clf.predict_proba(X_test)[:,1] > THRESHOLD, 1, 0)

pd.DataFrame(data=[accuracy_score(y_test, preds), recall_score(y_test, preds),
                   precision_score(y_test, preds), roc_auc_score(y_test, preds)],
             index=["accuracy", "recall", "precision", "roc_auc_score"])
```

Out[37]:

	0
accuracy	0.531646
recall	0.579618
precision	0.526012
roc_auc_score	0.531947

In [38]:

```
from sklearn import model_selection, preprocessing, linear_model, naive_bayes, metrics, 
svm
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn import decomposition, ensemble

import pandas, xgboost, numpy, textblob, string
from keras.preprocessing import text, sequence
from keras import layers, models, optimizers

def train_model(classifier, feature_vector_train, label, feature_vector_valid, is_neura
l_net=False):
    # fit the training dataset on the classifier
    classifier.fit(feature_vector_train, label)

    # predict the labels on validation dataset
    predictions = classifier.predict(feature_vector_valid)

    if is_neural_net:
        predictions = predictions.argmax(axis=-1)

    return metrics.accuracy_score(predictions, valid_y)

# load the dataset
#data = open('data/corpus').read()
#labels, texts = [], []
#for i, line in enumerate(data.split("\n")):
#    content = line.split()
#    labels.append(content[0])
#    texts.append(" ".join(content[1:]))

# create a dataframe using texts and lables
#trainDF = pandas.DataFrame()
#trainDF['text'] = texts
#trainDF['label'] = labels
```

In [39]:

```
X_df.head()
```

Out[39]:

	CallTrace
0	6 6 63 6 42 120 6 195 120 6 6 114 114 1 1 252 ...
1	54 175 120 175 175 3 175 175 120 175 120 175 1...
2	6 11 45 33 192 33 5 197 192 6 33 5 3 197 192 1...
3	7 174 174 5 197 197 6 13 195 4 4 118 6 91 38 5...
4	11 45 33 192 33 5 197 192 6 33 5 3 197 192 192...

In [40]:

```
# create a dataframe using texts and lables
trainDF = train_df.copy()

trainDF['CallTrace_T'] = trainDF.CallTrace.str.split(' ').str.join(',').astype(str)
#X_df = trainDF.drop('Label', axis=1)
X_df = trainDF.drop(['Label', 'CallTrace'], axis=1)
y = trainDF['Label']

# split the dataset into training and validation datasets
train_x, valid_x, train_y, valid_y = model_selection.train_test_split(X_df, y)

# label encode the target variable
encoder = preprocessing.LabelEncoder()
train_y = encoder.fit_transform(train_y)
valid_y = encoder.fit_transform(valid_y)

X_df.head()
#list(encoder.classes_)
#le_name_mapping = dict(zip(encoder.classes_, encoder.transform(encoder.classes_)))
#print(le_name_mapping)
```

Out[40]:

	CallTrace_T
0	6,6,63,6,42,120,6,195,120,6,6,114,114,1,1,252,...
1	54,175,120,175,175,3,175,175,120,175,120,175,1...
2	6,11,45,33,192,33,5,197,192,6,33,5,3,197,192,1...
3	7,174,174,5,197,197,6,13,195,4,4,118,6,91,38,5...
4	11,45,33,192,33,5,197,192,6,33,5,3,197,192,192...

In [41]:

```
train_x.shape, valid_x.shape, train_y.shape, valid_y.shape
```

Out[41]:

```
((1184, 1), (395, 1), (1184,), (395,))
```

In [42]:

```
trainDF.head()
```

Out[42]:

	Label	CallTrace	CallTrace_T
0	Normal	6 6 63 6 42 120 6 195 120 6 6 114 114 1 1 252 ...	6,6,63,6,42,120,6,195,120,6,6,114,114,1,1,252,...
1	Normal	54 175 120 175 175 3 175 175 120 175 120 175 1...	54,175,120,175,175,3,175,175,120,175,120,175,1...
2	Normal	6 11 45 33 192 33 5 197 192 6 33 5 3 197 192 1...	6,11,45,33,192,33,5,197,192,6,33,5,3,197,192,1...
3	Normal	7 174 174 5 197 197 6 13 195 4 4 118 6 91 38 5...	7,174,174,5,197,197,6,13,195,4,4,118,6,91,38,5...
4	Normal	11 45 33 192 33 5 197 192 6 33 5 3 197 192 192...	11,45,33,192,33,5,197,192,6,33,5,3,197,192,192...

Feature Engineering - 1n, 2n, 3n-grams

In [43]:

```
trainDF.head()
```

Out[43]:

	Label	CallTrace	CallTrace_T
0	Normal	6 6 63 6 42 120 6 195 120 6 6 114 114 1 1 252 ...	6,6,63,6,42,120,6,195,120,6,6,114,114,1,1,252,...
1	Normal	54 175 120 175 175 3 175 175 120 175 120 175 1...	54,175,120,175,175,3,175,175,120,175,120,175,1...
2	Normal	6 11 45 33 192 33 5 197 192 6 33 5 3 197 192 1...	6,11,45,33,192,33,5,197,192,6,33,5,3,197,192,1...
3	Normal	7 174 174 5 197 197 6 13 195 4 4 118 6 91 38 5...	7,174,174,5,197,197,6,13,195,4,4,118,6,91,38,5...
4	Normal	11 45 33 192 33 5 197 192 6 33 5 3 197 192 192...	11,45,33,192,33,5,197,192,6,33,5,3,197,192,192...

In [44]:

```
train_1n = pd.read_csv("./train_1n.csv")
train_1n.columns
train_1n_bkp = train_1n.copy()
train_1n.head()
```

Out[44]:

	Label	168	265	3	54	162	142	309	146	114	175	43	104	5	78	102	13	6
0	Adduser	193	75	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0
1	Adduser	0	110	139	0	0	286	0	55	0	64	0	50	0	0	0	0	0
2	Adduser	249	133	112	0	0	0	0	0	0	0	60	0	0	0	0	0	0
3	Adduser	0	1	51	809	0	0	202	0	0	0	0	0	0	0	0	0	0
4	Adduser	426	234	157	0	0	0	0	0	0	0	0	0	0	2	0	0	0

In [45]:

```
train_1n.columns
```

Out[45]:

```
Index(['Label', '168', '265', '3', '54', '162', '142', '309', '146', '11
4',
       '175', '43', '104', '5', '78', '102', '13', '6', '240', '4', '192',
       '195', '91', '85', '125', '197', '140', '19', '174', '301', '221',
'33',
       '180', '45', '196', '120', '7', '220', '42', '63', '11', '1', '25
2',
       '201', '243', '199', '308', '122', '118', '219'],
      dtype='object')
```

Modelling Logistic Regression - 1n-grams

In [46]:

```
import warnings
warnings.filterwarnings("ignore")

# split the dataset in train and test

#y = train_1n.iloc[:, 0].values
#train_1n_no_y = train_1n.drop('Label', axis=1)
#X = train_1n_no_y.iloc[:, :].values
y = train_1n.iloc[:, 0]
train_1n_no_y = train_1n.drop('Label', axis=1)
X = train_1n_no_y.iloc[:, :]


# Splitting the dataset into the Training set and Test set
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
= 100)
```

In [47]:

```
X_test_bkp = X_test
```

In [48]:

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape, type(X), type(y)
```

Out[48]:

```
((1070, 49),
 (268, 49),
 (1070,),
 (268,),
 pandas.core.frame.DataFrame,
 pandas.core.series.Series)
```

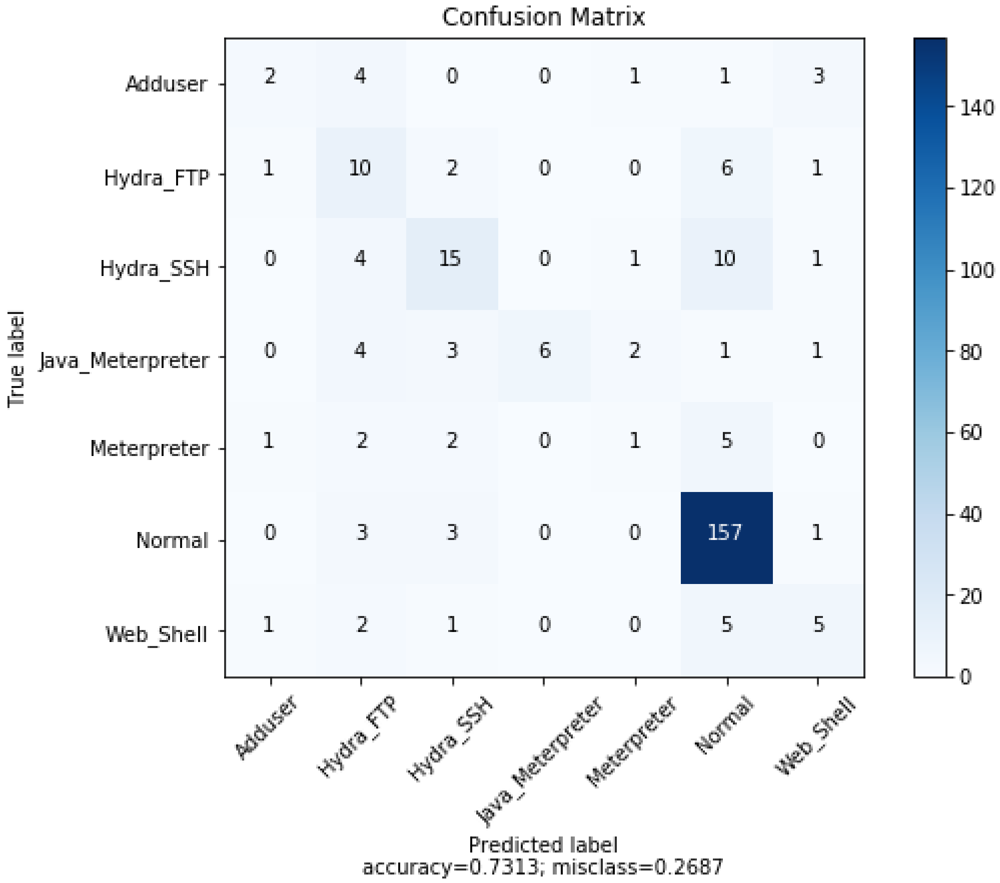
In [49]:

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(multi_class='ovr', solver = 'lbfgs')
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# How did our model perform?
from sklearn import metrics
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
```

```
Misclassified samples: 72
Accuracy: 0.73
```

In [50]:

```
#classifier.predict_proba(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cm,
                      normalize    = False,
                      target_names = ['Adduser', 'Hydra_FTP', 'Hydra_SSH', 'Java_Meterp
reter', 'Meterpreter', 'Normal', 'Web_Shell'],
                      title        = "Confusion Matrix")
```



In [51]:

```
y_pred.shape, y_test.shape, type(y_test)
```

Out[51]:

```
((268,), (268,), pandas.core.series.Series)
```

In [52]:

```
# Merge predicted results into original dataframe
# y_test['preds'] = y_pred
# df_out = pd.merge(train_1n, y_test[['preds']], how = 'left', right_index = True)
```

In [53]:

```
train_1n.index
```

Out[53]:

```
RangeIndex(start=0, stop=1338, step=1)
```

In [54]:

```
train_2n = pd.read_csv("./train_2n.csv")
train_2n.columns
train_2n_bkp = train_2n.copy()
train_2n.head()
```

Out[54]:

	Label	168 168	54 54	168 265	162 162	265 168	3 168	168 3	265 265	3 3	265 3	3 265	54 309	309 54	114 162	162 114	14 14
0	Adduser	138	0	48	0	47	0	0	24	0	0	0	0	0	0	0	
1	Adduser	0	0	0	0	0	0	0	24	45	17	20	0	0	0	0	12
2	Adduser	110	0	60	0	55	48	52	28	16	31	32	0	0	0	0	
3	Adduser	0	594	0	0	0	0	0	0	1	0	0	172	165	0	0	
4	Adduser	236	0	117	0	119	69	71	69	38	46	48	0	0	0	0	

In [55]:

```
train_3n = pd.read_csv("./train_3n.csv")
train_3n.columns
train_3n_bkp = train_3n.copy()
train_3n.head()
```

Out[55]:

	Label	168 168 168	54 54 54	162 162 162	168 265 168	265 168 168	168 168 265	168 3 168	168 168 3	3 168 168	54 309 54	54 54 309	265 168 265	309 54 54	168 265 265	265 265 168	1 1
0	Adduser	101	0	0	31	34	31	0	0	0	0	0	12	0	14	14	
1	Adduser	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	Adduser	49	0	0	25	26	25	22	23	21	0	0	12	0	11	14	
3	Adduser	0	431	0	0	0	0	0	0	0	137	128	0	124	0	0	
4	Adduser	132	0	0	63	68	60	33	42	36	0	0	32	0	32	31	

In [56]:

```
train_1n.shape, train_2n.shape, train_3n.shape
```

Out[56]:

```
((1338, 50), (1338, 800), (1338, 4148))
```

Modelling Logistic Regression/SVM/RandomForrest - 1n-grams + 2n-grams + 3n-grams

In [57]:

```
frames=[train_1n, train_2n, train_3n]
result=pd.concat(frames, axis=1)
result.shape
```

Out[57]:

```
(1338, 4998)
```

In [58]:

```
result.head()
```

Out[58]:

	Label	168	265	3	54	162	142	309	146	114	175	43	104	5	78	102	13	6
0	Adduser	193	75	0	0	0	0	0	0	0	0	0	0	0	0	10	0	0
1	Adduser	0	110	139	0	0	286	0	55	0	64	0	50	0	0	0	0	0
2	Adduser	249	133	112	0	0	0	0	0	0	0	60	0	0	0	0	0	0
3	Adduser	0	1	51	809	0	0	202	0	0	0	0	0	0	0	0	0	0
4	Adduser	426	234	157	0	0	0	0	0	0	0	0	0	0	2	0	0	0

In [59]:

```
result.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Columns: 4998 entries, Label to 85 196 5
dtypes: int64(4995), object(3)
memory usage: 51.0+ MB
```

In [60]:

```
import warnings
warnings.filterwarnings("ignore")

# split the dataset in train and test

y = result.iloc[:, 0].values
result_no_y = result.drop('Label', axis=1)
X = result_no_y.iloc[:, :].values
```

In [61]:

```
#result
```

In [62]:

```
# Splitting the dataset into the Training set and Test set
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
= 0)
X_train.shape, X_test.shape, y_train.shape, y_test.shape, type(X), type(y)
```

Out[62]:

```
((1070, 4995), (268, 4995), (1070,), (268,), numpy.ndarray, numpy.ndarray)
```

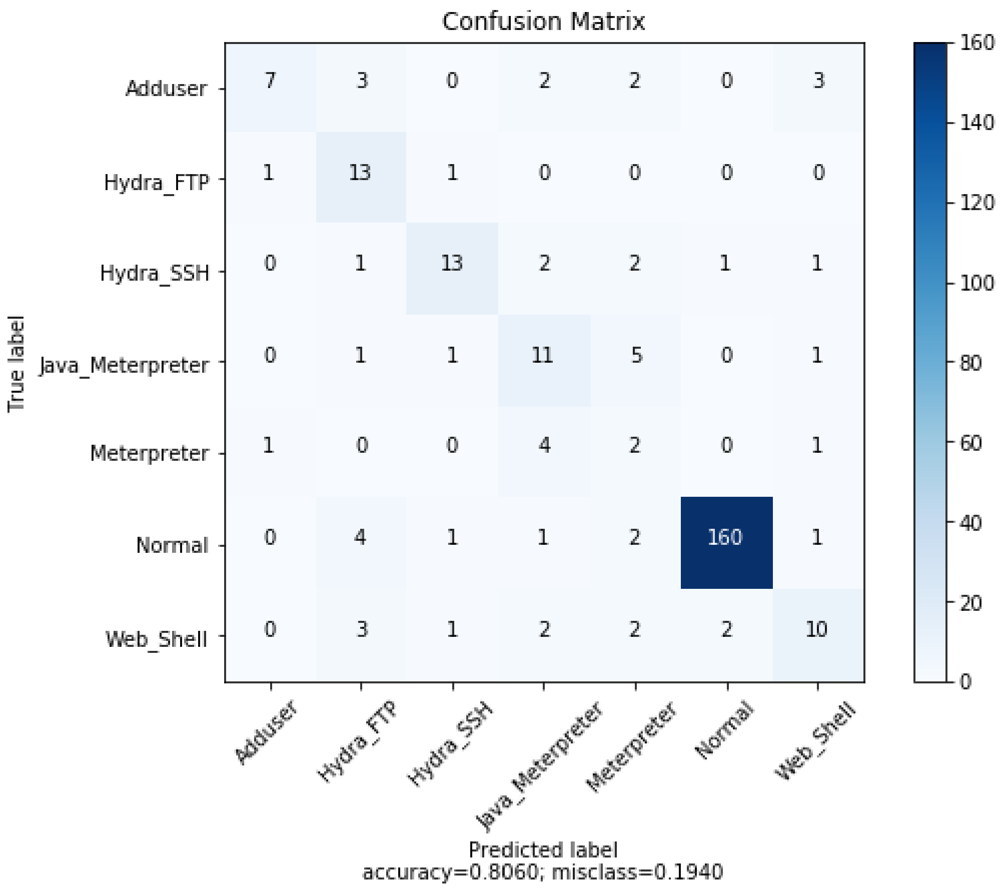
In [63]:

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
#classifier = LogisticRegression(multi_class='ovr', solver = 'lbfgs')
#classifier = SVC(kernel = 'linear', random_state = 0)
#classifier = SVC(kernel = 'rbf', random_state = 0)
clf = RandomForestClassifier(n_jobs=-1)
classifier.fit(X_train, y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# How did our model perform?
from sklearn import metrics
count_misclassified = (y_test != y_pred).sum()
print('Misclassified samples: {}'.format(count_misclassified))
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
#y_pred
```

```
Misclassified samples: 52
Accuracy: 0.81
```

In [64]:

```
#classifier.predict_proba(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cm,
                      normalize    = False,
                      target_names = ['Adduser', 'Hydra_FTP', 'Hydra_SSH', 'Java_Meterp
reter', 'Meterpreter', 'Normal', 'Web_Shell'],
                      title        = "Confusion Matrix")
```



Applying 10-Fold cross-validation

In [65]:

```
from sklearn.model_selection import cross_val_score
import numpy as np

print(np.mean(cross_val_score(clf, X_train, y_train, cv=10)))
```

```
0.806636559725912
```

Comparing Different Models BinaryCalssification - 1n-grams + 2n-grams + 3n-grams

In [66]:

```
# Compare Algorithms
# https://machinelearningmastery.com/compare-machine-learning-algorithms-python-scikit-
learn/

import pandas
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

# load dataset
Y = result.iloc[:, 0].values
result_no_y = result.drop('Label', axis=1)
X = result_no_y.iloc[:, :].values
#X = array[:,0:8]
#Y = array[:,8]

# Prepare configuration for cross validation test harness
seed = 7
```

In [67]:

```
# Prepare models
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
models.append(('RandomForest', RandomForestClassifier()))
```

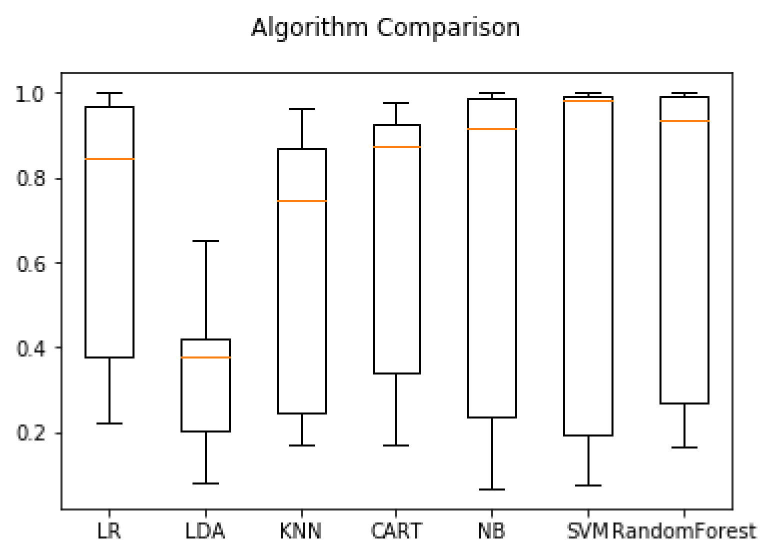
In [68]:

```
# Evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
        kfold = model_selection.KFold(n_splits=10, random_state=seed)
        cv_results = model_selection.cross_val_score(model, X, Y, cv=kfold, scoring=sco
ring)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
        print(msg)
```

```
LR: 0.702951 (0.299686)
LDA: 0.351554 (0.183791)
KNN: 0.601403 (0.316366)
CART: 0.669336 (0.316557)
NB: 0.647756 (0.392859)
SVM: 0.655218 (0.410400)
RandomForest: 0.679845 (0.366389)
```

In [69]:

```
# Boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.show()
```



Random Forest & Logistic Regression gave best accuracy so far

Random Forest Model Parameter Tuning

In [70]:

```
#frames = [train_1n, train_2n, train_3n]
frames = [train_1n, train_2n, train_3n]
result = pd.concat(frames, axis=1)

result = result.loc[:,~result.columns.duplicated()]

result.loc[result.Label != 'Normal','Label']= 1
result.loc[result.Label == 'Normal','Label']= 0
result.head()
#result['Label_Binary'].value_counts()

# Extract features and labels
labels = result['Label']
features = result.drop('Label', axis = 1)
#features.head(5)
#labels.head(5)


# One Hot Encoding
#features = pd.get_dummies(result)
#features.head(5)


#from sklearn import preprocessing

#le = preprocessing.LabelEncoder()
#features['Label_Normal'] = le.fit_transform(features['Label_Normal'])


#cols_drop = [ 'Label_Meterpreter', 'Label_Web_Shell', 'Label_Adduser',
#       'Label_Hydra_FTP', 'Label_Hydra_SSH', 'Label_Java_Meterpreter', 'Label_Normal']

# Extract features and labels
#labels = features['Label_Normal']
#labels['Label_Normal'].astype(object).astype(int)
#labels = labels.loc[:,~labels.columns.duplicated()]
#features = features.drop(cols_drop, axis = 1)
#features.head(5)
#labels.head(5)
```

In [71]:

```
# Convert to numpy arrays
import numpy as np

features = np.array(features)
labels = np.array(labels)

# Training and Testing Sets
from sklearn.model_selection import train_test_split

train_features, test_features, train_labels, test_labels = train_test_split(features, l
abels,
                                                                              test_size =
0.25, random_state = 42)
```

In [72]:

```
print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)
```

```
Training Features Shape: (1003, 4995)
Training Labels Shape: (1003,)
Testing Features Shape: (335, 4995)
Testing Labels Shape: (335,)
```

Examine the Default Random Forest to Determine Parameters

In [73]:

```
# Reference : https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in
-python-using-scikit-learn-28d2aa77dd74

from sklearn.ensemble import RandomForestClassifier
from pprint import pprint

rf = RandomForestClassifier(random_state=42)

#Look at parameters used by  our current forest
pprint(rf.get_params())
```

```
{'bootstrap': True,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 'warn',
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

Random Search with Cross Validation

In [74]:

```
from sklearn.model_selection import RandomizedSearchCV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]

# Create the random grid
random_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,
               'bootstrap': bootstrap}

pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000]}
```

In [75]:

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestClassifier(random_state = 42)
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator=rf, param_distributions=random_grid,
                              n_iter = 100, scoring='accuracy',
                              cv = 3, verbose=2, random_state=42, n_jobs=-1,
                              return_train_score=True)

# Fit the random search model
rf_random.fit(train_features, train_labels);
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   49.6s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:  4.3min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  8.6min finished
```

Best Parameters Identified

In [76]:

```
rf_random.best_params_
```

Out[76]:

```
{'n_estimators': 600,
 'min_samples_split': 5,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': 60,
 'bootstrap': False}
```

In [77]:

```
#rf_random.cv_results_
```

Evaluate Random Search

To determine if random search yielded a better model, we compare the base model with the best random search model.

In [78]:

```
def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    accuracy = metrics.accuracy_score(test_labels, predictions)
    print('Accuracy: {:.2f}'.format(accuracy))
```

Evaluate the Default Model

In [79]:

```
base_model = RandomForestClassifier(n_estimators = 10, random_state = 42)
base_model.fit(train_features, train_labels)
base_accuracy = evaluate(base_model, test_features, test_labels)
```

```
Accuracy: 0.93
```

Evaluate the Best Random Search Model

In [80]:

```
best_random = rf_random.best_estimator_
random_accuracy = evaluate(best_random, test_features, test_labels)
```

```
Accuracy: 0.96
```

Grid Search

We can now perform grid search building on the result from the random search. We will test a range of hyperparameters around the best values returned by random search.

In [81]:

```
from sklearn.model_selection import GridSearchCV

# Create the parameter grid based on the results of random search
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [2, 3],
    'min_samples_leaf': [3, 4, 5],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300, 1000]
}

# Create a base model
rf = RandomForestClassifier(random_state = 42)

# Instantiate the grid search model
grid_search = GridSearchCV(estimator = rf, param_grid = param_grid,
                          cv = 3, n_jobs = -1, verbose = 2, return_train_score=True)
```

In [82]:

```
# Fit the grid search to the data
grid_search.fit(train_features, train_labels);
```

```
Fitting 3 folds for each of 288 candidates, totalling 864 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent worker
s.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    2.3s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   13.2s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:   31.9s
[Parallel(n_jobs=-1)]: Done 632 tasks      | elapsed:   58.2s
[Parallel(n_jobs=-1)]: Done 864 out of 864 | elapsed:  1.3min finished
```

In [83]:

```
grid_search.best_params_
```

Out[83]:

```
{'bootstrap': True,
 'max_depth': 80,
 'max_features': 3,
 'min_samples_leaf': 3,
 'min_samples_split': 8,
 'n_estimators': 1000}
```

Evaluate the Best Model from Grid Search

In [84]:

```
best_grid = grid_search.best_estimator_
grid_accuracy = evaluate(best_grid, test_features, test_labels)
```

```
Accuracy: 0.78
```

Final Model

In [85]:

```
final_model = grid_search.best_estimator_

print('Final Model Parameters:\n')
pprint(final_model.get_params())
print('\n')
grid_final_accuracy = evaluate(final_model, test_features, test_labels)
```

```
Final Model Parameters:

{'bootstrap': True,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': 80,
 'max_features': 3,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 3,
 'min_samples_split': 8,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 1000,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}


Accuracy: 0.78
```