Branch: master

Find file Copy path

Springboard / Capstone-2 / systemcall-anomaly-detection / **logreg_on_adfa_ld.ipynb**

Kumud Nawani Capstone-2 ML

191c964 3 minutes ago

1 contributor

Download History

2.19 MB

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 RandomForestRegresso
r, RandomForestClassifier
from IPython.display import display
from sklearn import metrics
from sklearn.model_selection import train_test_sp
lit
import statistics
import numpy as np
from sklearn import metrics
from sklearn.preprocessing import MinMaxScaler, S
tandardScaler, LabelEncoder
from sklearn.feature_selection import SelectKBest
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_sp
lit, 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_sequ
ences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM,
SpatialDropout1D
from sklearn.model_selection import train_test_sp
lit
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 matri
x',
                          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, rota
tion=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, misclass))
    plt.show()

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

```
ange(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_Shell']

# initializing dictionary for n-grams from all fi
les
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 infi
le:
                                globals()['%s_lis
t_array' % term+str(k)] = ALine =infile.read()
                                #ALine = ALine[:8
20]
                                Attack_list_new.a
ppend(term +','+ str(ALine))
                                globals()['%s_lis
t' % term].extend(globals()['%s_list_array' % ter
m+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("                 Extractin
g 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_Maste
r/"
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()['Nor
mal%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 create
d...............................................\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 create
d...............................................
```

In [4]:

```
new_train_list = []
new_train_list = Normal_list_new + Attack_list_ne
w
```

```
#new_train_list[1]
#Attack_list_new[1]
```

In [5]:

```
new_train_list = []
new_train_list = Normal_list_new + Attack_list_ne
w


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=',',er
ror_bad_lines=False, header=None, names=['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('c
ategory')

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]:

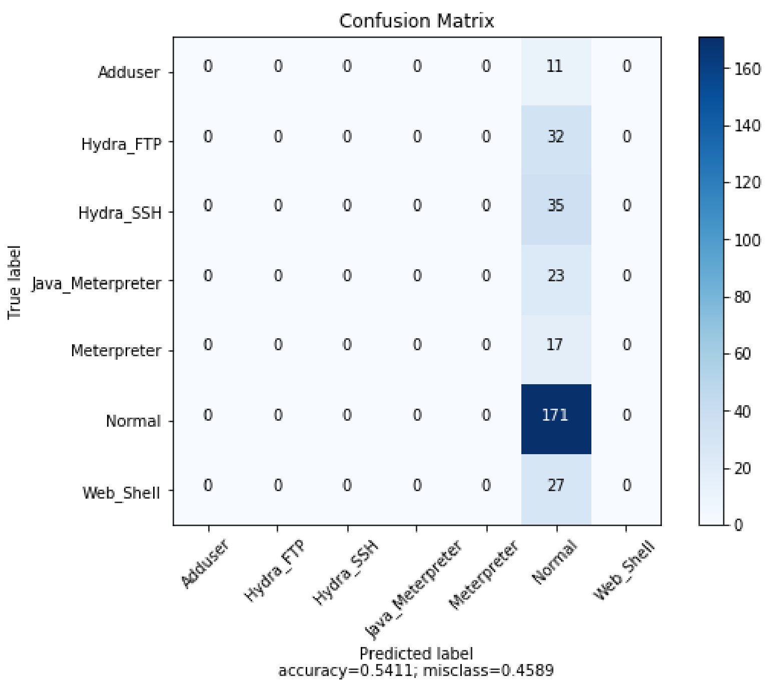
```
import warnings
warnings.filterwarnings("ignore")

# 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_sp
lit
X_train, X_test, y_train, y_test = train_test_spl
it(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 LogisticRegressi
on
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_mi
sclassified))
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_Meterpreter', 'Me
terpreter', 'Normal', 'Web_Shell'],
                      title        = "Confusion M
atrix")
```



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	Lab
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

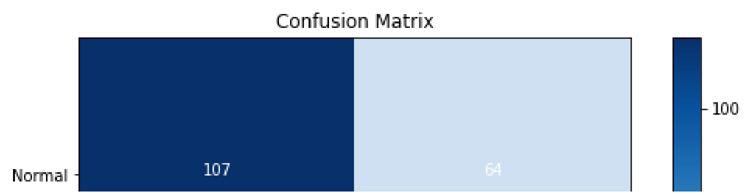
In [13]:

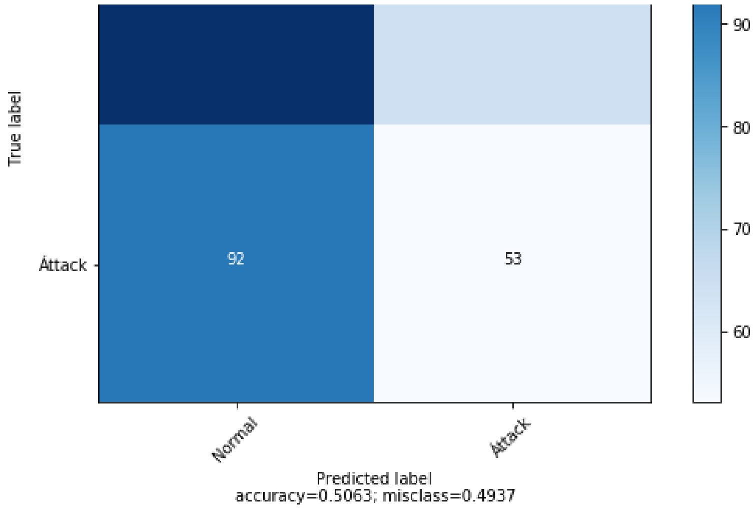
```
import warnings
warnings.filterwarnings("ignore")

# 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_sp
lit
X_train, X_test, y_train, y_test = train_test_spl
it(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 LogisticRegressi
on
#classifier = LogisticRegression(multi_class='ov
r', 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_mi
sclassified))
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 M
atrix")
```





In [15]:

```
print(metrics.classification_report(y_pred, y_tes
t))
```

```
              precision    recall  f1-score   supp
ort

         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_transfo
rm(np.array(label_df[col].astype(str)).reshape((-
1,)))
        mappings.append(dict(zip(label_encoder.cl
asses_, range(1, len(label_encoder.classes_)+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_transf
orm(np.array(label_df[col].astype(str)).reshape((
-1,)))
        mappings.append(dict(zip(onehot_encoder.c
lasses_, range(1, len(onehot_encoder.classes_)+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_spl
it(label_df, y, test_size = 0.2, random_state = 1
0)
X_train.shape, X_test.shape, y_train.shape, y_tes
t.shape
```

Out[24]:

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]:

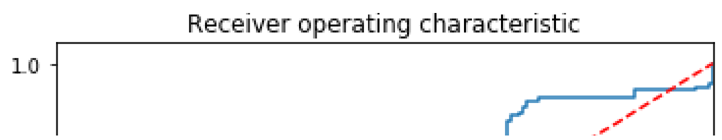
```
LogisticRegression(C=1.0, class_weight=None, dual=
False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi
_class='warn',
          n_jobs=None, penalty='l2', random_state=
None, solver='warn',
          tol=0.0001, verbose=0, warm_start=False)
```

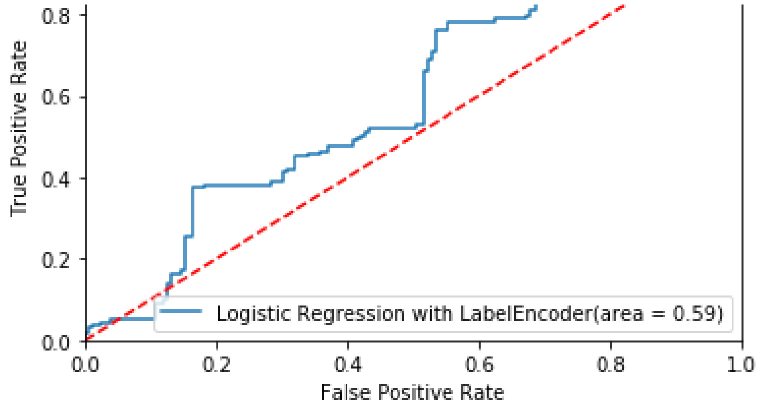
In [27]:

```
# logit_roc_auc = roc_auc_score(y_test, model_nor
m.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]:

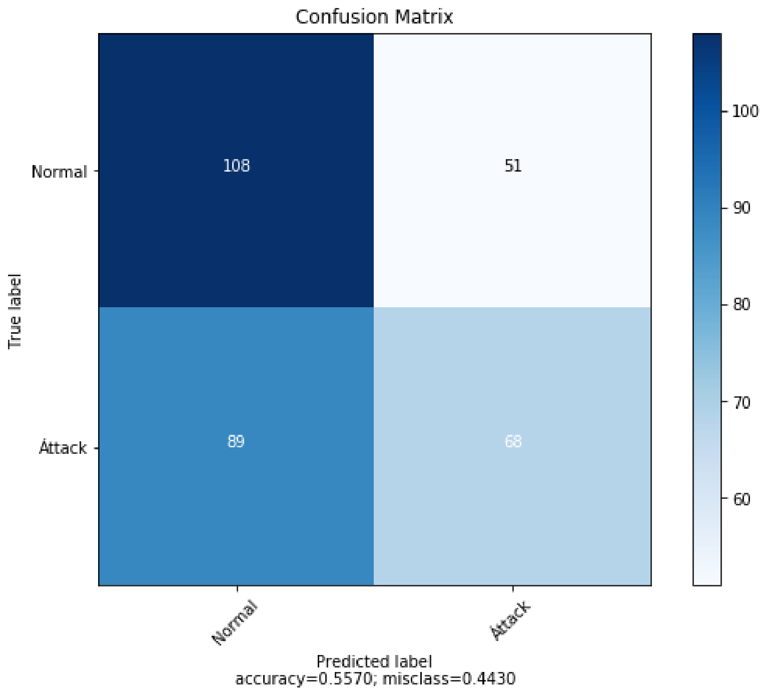
```
labelfpr, labeltpr, labelthreshold = metrics.roc_
curve(y_test, preds)
label_roc_auc = metrics.auc(labelfpr, labeltpr)

plt.figure()
plt.plot(labelfpr, labeltpr, label='Logistic Regr
ession with LabelEncoder(area = %0.2f)' % label_r
oc_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 M
atrix")
```



In [30]:

```
X_train.shape, X_test.shape, y_train.shape, y_tes
t.shape
```

Out[30]:

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

In [31]:

```
print(metrics.classification_report(classes, y_te
st))
```

```
              precision    recall  f1-score   supp
ort
```

```
           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 value
s to between 0 and 1.
from sklearn.preprocessing import MinMaxScaler, S
tandardScaler

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

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

In [33]:

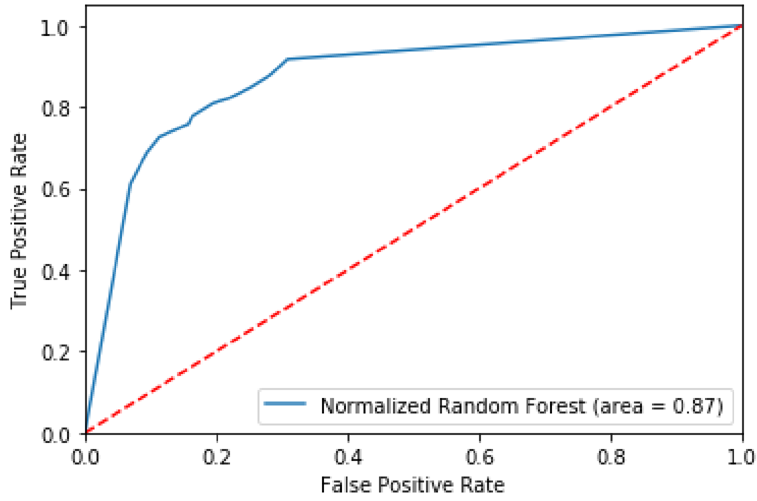
```
clf = RandomForestClassifier(n_jobs=-1)
model_rf = clf.fit(X_train, y_train)
print('Model accuracy is',model_rf.score(X_test,
y_test))
```

```
Model accuracy is 0.8069620253164557
```

In [34]:

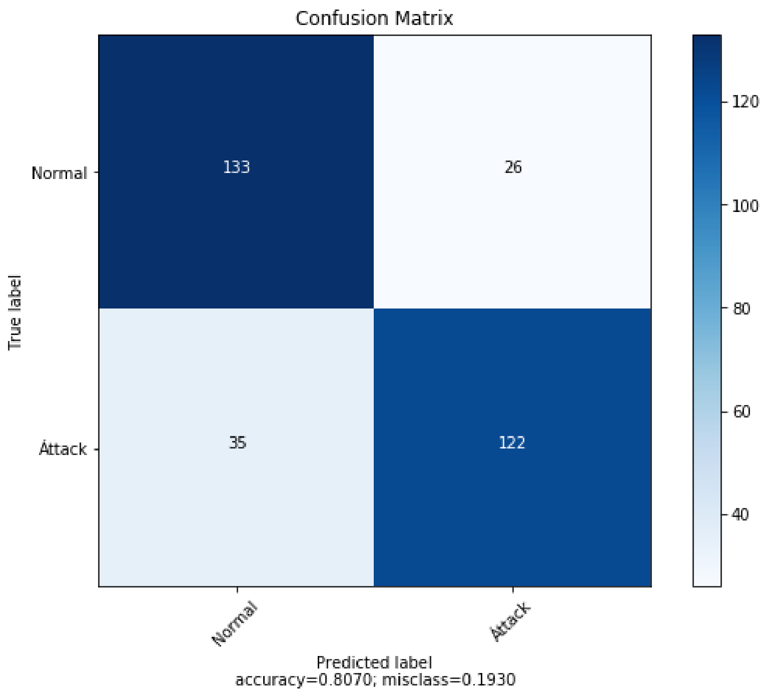
```
probs = model_rf.predict_proba(X_test)
preds = probs[:,1]
rffpr, rftpr, rfthreshold = metrics.roc_curve(y_t
est, preds)
rf_roc_auc = metrics.auc(rffpr, rftpr)

plt.figure()
plt.plot(rffpr, rftpr, label='Normalized Random F
orest (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()
```

Receiver operating characteristic



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 M
atrix")
```



In [36]:

```
print(metrics.classification_report(classes, y_te
st))
```

```
              precision    recall  f1-score   supp
ort

           0       0.84      0.79      0.81
168
           1       0.78      0.82      0.80
148
```

```
   micro avg       0.81      0.81      0.81
316
   macro avg       0.81      0.81      0.81
316
weighted avg       0.81      0.81      0.81
316
```

Train Data with ngrams

In [37]:

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_sp
lit
from sklearn.linear_model import LogisticRegressi
on
from sklearn.metrics import accuracy_score, confu
sion_matrix, recall_score, roc_auc_score, precisi
on_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=1
0
)

#X_train, X_test, y_train, y_test = train_test_sp
lit(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", "precis
ion", "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, preprocessin
g, linear_model, naive_bayes, metrics, svm
```

```
g, linear_model, naive_bayes, metrics, svm
from sklearn.feature_extraction.text import Tfidf
Vectorizer, 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_neural_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_vect
or_valid)

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

    return metrics.accuracy_score(predictions, va
lid_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.sp
lit(' ').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	Normal	54 175 120 175 175 3 175 175 120	54,175,120,175,175,3,175,175,120,175

		175 120 175 1...	
2	Normal	6 11 45 33 192 33 5 197 192 6 33 5 3 197 192 1...	6,11,45,33,192,33,5,197,192,6,33,5,3,1
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
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

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	Normal	54 175 120 175 175 3 175 175 120 175 120 175 1...	54,175,120,175,175,3,175,175,120,175
2	Normal	6 11 45 33 192 33 5 197 192 6 33 5 3 197 192 1...	6,11,45,33,192,33,5,197,192,6,33,5,3,1
		7 174 174 5 197 197	

3	Normal	6 13 195 4 4 118 6 91 38 5...	7,174,174,5,197,197,6,13,195,4,4,118,6
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

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	17
0	Adduser	193	75	0	0	0	0	0	0	0	0
1	Adduser	0	110	139	0	0	286	0	55	0	64
2	Adduser	249	133	112	0	0	0	0	0	0	0
3	Adduser	0	1	51	809	0	0	202	0	0	0
4	Adduser	426	234	157	0	0	0	0	0	0	0

In [45]:

```
train_1n.columns
```

Out[45]:

```
Index(['Label', '168', '265', '3', '54', '162', '1
42', '309', '146', '114',
       '175', '43', '104', '5', '78', '102', '13',
'6', '240', '4', '192',
       '195', '91', '85', '125', '197', '140', '1
9', '174', '301', '221', '33',
       '180', '45', '196', '120', '7', '220', '4
2', '63', '11', '1', '252',
       '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)
```

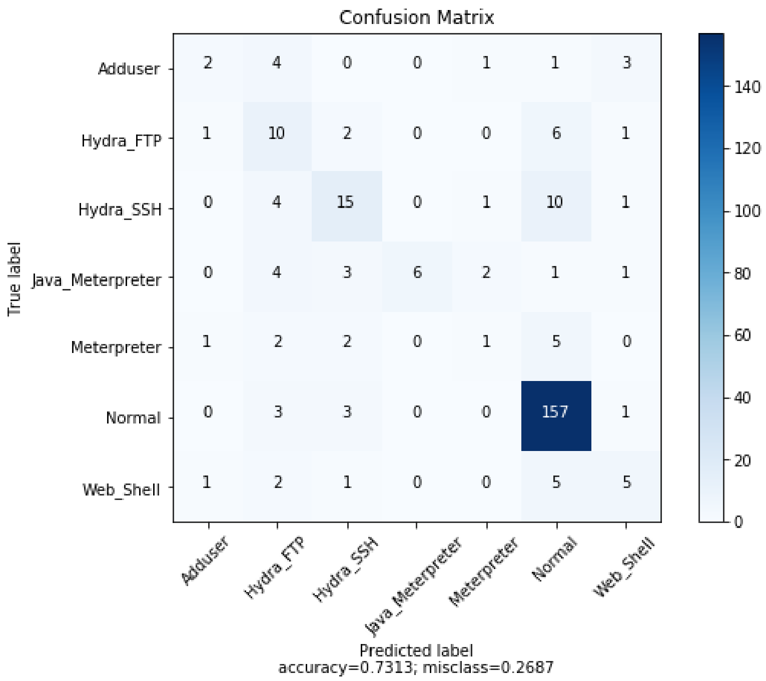
In [49]:

```
# 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_Meterpreter', 'Me
terpreter', 'Normal', 'Web_Shell'],
                      title        = "Confusion M
atrix")
```



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
0	Adduser	138	0	48	0	47	0	0	24	0	0
1	Adduser	0	0	0	0	0	0	0	24	45	17
2	Adduser	110	0	60	0	55	48	52	28	16	31
3	Adduser	0	594	0	0	0	0	0	0	1	0
4	Adduser	236	0	117	0	119	69	71	69	38	46

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 30 54
0	Adduser	101	0	0	31	34	31	0	0	0	0
1	Adduser	0	0	0	0	0	0	0	0	0	0
2	Adduser	49	0	0	25	26	25	22	23	21	0
3	Adduser	0	431	0	0	0	0	0	0	0	13
4	Adduser	132	0	0	63	68	60	33	42	36	0

In [56]:

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

Out[56]:

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

In [70]:

```
train_1n_30 = train_1n.iloc[:, 0:30]
train_2n_30 = train_2n.iloc[:, 0:30]
train_3n_30 = train_3n.iloc[:, 0:30]
```

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

In [71]:

```
frames=[train_1n_30, train_2n_30, train_3n_30]
result=pd.concat(frames, axis=1)
result.shape
```

Out[71]:

```
(1338, 90)
```

In [72]:

```
result.head()
```

Out[72]:

	Label	168	265	3	54	162	142	309	146	114	17
0	Adduser	193	75	0	0	0	0	0	0	0	0
1	Adduser	0	110	139	0	0	286	0	55	0	64
2	Adduser	249	133	112	0	0	0	0	0	0	0
3	Adduser	0	1	51	809	0	0	202	0	0	0
4	Adduser	426	234	157	0	0	0	0	0	0	0

In [73]:

```
result.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 90 columns):
Label          1338 non-null object
168            1338 non-null int64
265            1338 non-null int64
3              1338 non-null int64
54             1338 non-null int64
162            1338 non-null int64
142            1338 non-null int64
309            1338 non-null int64
146            1338 non-null int64
114            1338 non-null int64
175            1338 non-null int64
43             1338 non-null int64
104            1338 non-null int64
5              1338 non-null int64
78             1338 non-null int64
102            1338 non-null int64
13             1338 non-null int64
6              1338 non-null int64
240            1338 non-null int64
4              1338 non-null int64
192            1338 non-null int64
195            1338 non-null int64
91             1338 non-null int64
85             1338 non-null int64
125            1338 non-null int64
197            1338 non-null int64
```

```
140            1338 non-null int64
19             1338 non-null int64
174            1338 non-null int64
301            1338 non-null int64
Label          1338 non-null object
168 168        1338 non-null int64
54 54          1338 non-null int64
168 265        1338 non-null int64
162 162        1338 non-null int64
265 168        1338 non-null int64
3 168          1338 non-null int64
168 3          1338 non-null int64
265 265        1338 non-null int64
3 3            1338 non-null int64
265 3          1338 non-null int64
3 265          1338 non-null int64
54 309         1338 non-null int64
309 54         1338 non-null int64
114 162        1338 non-null int64
162 114        1338 non-null int64
142 142        1338 non-null int64
142 3          1338 non-null int64
3 142          1338 non-null int64
142 265        1338 non-null int64
265 142        1338 non-null int64
3 54           1338 non-null int64
174 174        1338 non-null int64
309 309        1338 non-null int64
43 168         1338 non-null int64
168 146        1338 non-null int64
142 146        1338 non-null int64
146 3          1338 non-null int64
146 142        1338 non-null int64
175 175        1338 non-null int64
Label          1338 non-null object
168 168 168    1338 non-null int64
54 54 54       1338 non-null int64
162 162 162    1338 non-null int64
168 265 168    1338 non-null int64
265 168 168    1338 non-null int64
168 168 265    1338 non-null int64
168 3 168      1338 non-null int64
168 168 3      1338 non-null int64
3 168 168      1338 non-null int64
54 309 54      1338 non-null int64
54 54 309      1338 non-null int64
265 168 265    1338 non-null int64
309 54 54      1338 non-null int64
168 265 265    1338 non-null int64
265 265 168    1338 non-null int64
162 114 162    1338 non-null int64
114 162 162    1338 non-null int64
162 162 114    1338 non-null int64
3 168 265      1338 non-null int64
168 265 3      1338 non-null int64
265 3 168      1338 non-null int64
3 265 168      1338 non-null int64
265 168 3      1338 non-null int64
168 3 265      1338 non-null int64
265 265 265    1338 non-null int64
3 168 3        1338 non-null int64
```

```
3 3 168        1338 non-null int64
168 3 3        1338 non-null int64
3 3 3          1338 non-null int64
dtypes: int64(87), object(3)
memory usage: 940.9+ KB
```

In [74]:

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

# split the dataset in train and test
result = result.loc[:,~result.columns.duplicated
()]

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

In [75]:

```
#result
```

In [76]:

```
# Splitting the dataset into the Training set and
Test set
from sklearn.model_selection import train_test_sp
lit
X_train, X_test, y_train, y_test = train_test_spl
it(X, y, test_size = 0.2, random_state = 0)
X_train.shape, X_test.shape, y_train.shape, y_tes
t.shape, type(X), type(y)
```

Out[76]:

```
((1070, 87), (268, 87), (1070,), (268,), numpy.nda
rray, numpy.ndarray)
```

In [77]:

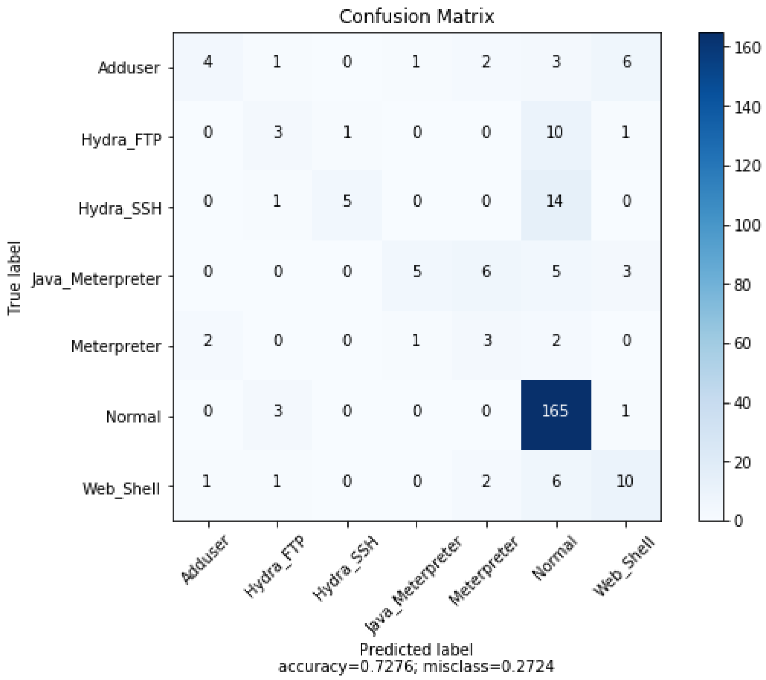
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC

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 LogisticRegressi
on
#classifier = LogisticRegression(multi_class='ov
r', 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_mi
sclassified))
accuracy = metrics.accuracy_score(y_test, y_pred)
print('Accuracy: {:.2f}'.format(accuracy))
#y_pred
```

```
Misclassified samples: 73
Accuracy: 0.73
```

In [78]:

```
#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_Meterpreter', 'Me
terpreter', 'Normal', 'Web_Shell'],
                      title        = "Confusion M
atrix")
```



Applying 10-Fold cross-validation

In [79]:

```
from sklearn.model_selection import cross_val_sco
re
import numpy as np

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

0.8121539825388545

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

In [80]:

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

import pandas
import matplotlib.pyplot as plt
from sklearn import model_selection
from sklearn.linear_model import LogisticRegressi
on
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifie
r
from sklearn.discriminant_analysis import LinearD
iscriminantAnalysis
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 [81]:

```
# 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', RandomForestClassi
fier()))
```

In [82]:

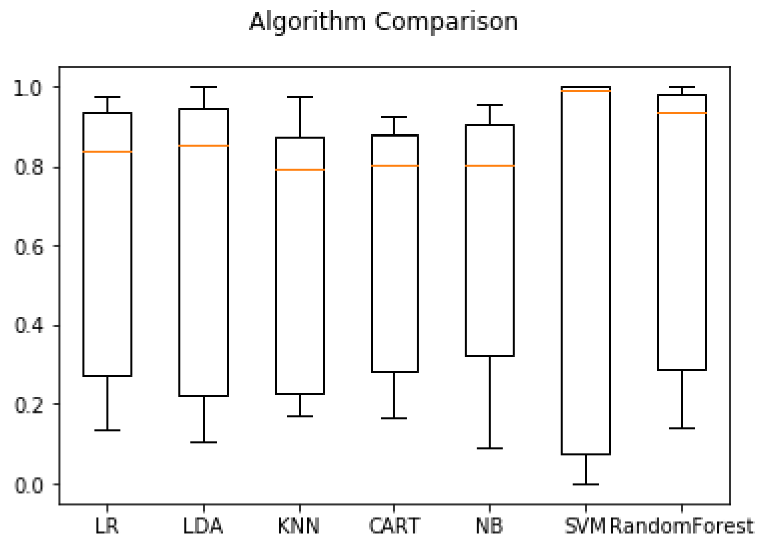
```
# Evaluate each model in turn
results = []
names = []
scoring = 'accuracy'
for name, model in models:
        kfold = model_selection.KFold(n_splits=10
, random_state=seed)
```

```
        cv_results = model_selection.cross_val_sc
ore(model, X, Y, cv=kfold, scoring=scoring)
        results.append(cv_results)
        names.append(name)
        msg = "%s: %f (%f)" % (name, cv_results.m
ean(), cv_results.std())
        print(msg)
```

```
LR: 0.629806 (0.349178)
LDA: 0.626826 (0.370299)
KNN: 0.602828 (0.329431)
CART: 0.616250 (0.311622)
NB: 0.616266 (0.331033)
SVM: 0.625373 (0.459750)
RandomForest: 0.675362 (0.364782)
```

In [83]:

```
# 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 [84]:

```
train_1n_30 = train_1n.iloc[:, 0:30]
train_2n_30 = train_2n.iloc[:, 0:30]
train_3n_30 = train_3n.iloc[:, 0:30]
```

In [85]:

```
train_1n_30.shape
```

Out[85]:

(1338, 30)

In [86]:

```
train_2n_30.shape
```

Out[86]:

(1338, 30)

In [87]:

```
train_3n_30.shape
```

Out[87]:

(1338, 30)

In [88]:

```
#frames = [train_1n, train_2n, train_3n]
frames = [train_1n_30, train_2n_30, train_3n_30]
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(feat
ures['Label_Normal'])


#cols_drop = [ 'Label_Meterpreter', 'Label_Web_Sh
ell', 'Label_Adduser',
#       'Label_Hydra_FTP', 'Label_Hydra_SSH', 'La
bel_Java_Meterpreter', 'Label_Normal']

# Extract features and labels
#labels = features['Label_Normal']
#labels['Label_Normal'].astype(object).astype(in
t)
#labels = labels.loc[:,~labels.columns.duplicated
()]
#features = features.drop(cols_drop, axis = 1)
```

```
#features.head(5)
#labels.head(5)
```

In [89]:

```
# 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_sp
lit

train_features, test_features, train_labels, test
_labels = train_test_split(features, labels,

test_size = 0.25, random_state = 42)
```

In [90]:

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

```
Training Features Shape: (1003, 87)
Training Labels Shape: (1003,)
Testing Features Shape: (335, 87)
Testing Labels Shape: (335,)
```

Examine the Default Random Forest to Determine Parameters

In [91]:

```
# Reference : https://towardsdatascience.com/hype
rparameter-tuning-the-random-forest-in-python-usi
ng-scikit-learn-28d2aa77dd74

from sklearn.ensemble import RandomForestClassifi
er
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 [92]:

```
from sklearn.model_selection import RandomizedSea
rchCV

# 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 n
ode
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 t
ree
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_s
plit,
               'min_samples_leaf': min_samples_le
af,
               '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 [93]:

```
# Use the random grid to search for best hyperpar
```

```
ameters
# 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 workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:   10.1s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   47.8s
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  1.6min finished
```

Best Parameters Identified

In [94]:

```
rf_random.best_params_
```

Out[94]:

```
{'n_estimators': 400,
 'min_samples_split': 2,
 'min_samples_leaf': 1,
 'max_features': 'sqrt',
 'max_depth': None,
 'bootstrap': False}
```

In [95]:

```
#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 [96]:

```
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 [97]:

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

```
Accuracy: 0.96
```

Evaluate the Best Random Search Model

In [98]:

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

```
Accuracy: 0.97
```

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 [99]:

```
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, ve
rbose = 2, return_train_score=True)
```

In [100]:

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

```
Fitting 3 folds for each of 288 candidates, totall
```

```
ing 864 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend w
ith 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elaps
ed:    2.5s
[Parallel(n_jobs=-1)]: Done 177 tasks      | elaps
ed:   14.0s
[Parallel(n_jobs=-1)]: Done 380 tasks      | elaps
ed:   29.6s
[Parallel(n_jobs=-1)]: Done 663 tasks      | elaps
ed:   51.8s
[Parallel(n_jobs=-1)]: Done 864 out of 864 | elaps
ed:  1.1min finished
```

In [101]:

```
grid_search.best_params_
```

Out[101]:

```
{'bootstrap': True,
 'max_depth': 80,
 'max_features': 3,
 'min_samples_leaf': 3,
 'min_samples_split': 8,
 'n_estimators': 300}
```

Evaluate the Best Model from Grid Search

In [102]:

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

```
Accuracy: 0.91
```

Final Model

In [103]:

```
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
```

```
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 300,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

```
Accuracy: 0.91
```

Deep Autoencoder Model

In [104]:

```
# https://www.kaggle.com/kredy10/simple-lstm-for-
text-classification
# https://www.curiousily.com/posts/credit-card-fr
aud-detection-using-autoencoders-in-keras/
```

In [107]:

```
import pandas as pd
import numpy as np
import pickle
import matplotlib.pyplot as plt
from scipy import stats
import tensorflow as tf
import seaborn as sns
from pylab import rcParams
from sklearn.model_selection import train_test_sp
lit
from keras.models import Model, load_model
from keras.layers import Input, Dense
from keras.callbacks import ModelCheckpoint, Tens
orBoard
from keras import regularizers

%matplotlib inline

sns.set(style='whitegrid', palette='muted', font_
scale=1.5)

rcParams['figure.figsize'] = 14, 8

RANDOM_SEED = 42
LABELS = ["Normal", "Attack"]
```

In [109]:

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

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

Out[109]:

	Label	168	265	3	54	162	142	309	146	114	175

0	1	193	75	0	0	0	0	0	0	0	0
1	1	0	110	139	0	0	286	0	55	0	64
2	1	249	133	112	0	0	0	0	0	0	0
3	1	0	1	51	809	0	0	202	0	0	0
4	1	426	234	157	0	0	0	0	0	0	0

In [110]:

```
attack = result[result.Label == 1]
normal = result[result.Label == 0]
```

In [111]:

```
attack.shape
```

Out[111]:

(505, 88)

In [112]:

```
normal.shape
```

Out[112]:

(833, 88)

In [ ]:

```

```

In [113]:

```
X_train, X_test = train_test_split(result, test_s
ize=0.2, random_state=RANDOM_SEED)
X_train = X_train[X_train.Label == 0]
X_train = X_train.drop(['Label'], axis=1)

y_test = X_test['Label']
X_test = X_test.drop(['Label'], axis=1)

X_train = X_train.values
X_test = X_test.values
```

In [114]:

```
X_train.shape
```

Out[114]:

(662, 87)

Building the model Our Autoencoder uses 4 fully connected layers with 14, 7, 7 and 29 neurons respectively. The first two layers are used for our encoder, the last two go for the decoder. Additionally, L1 regularization will be used during training:

training.

In [115]:

```
input_dim = X_train.shape[1]
encoding_dim = 341
```

In [116]:

```
input_layer = Input(shape=(input_dim, ))

encoder = Dense(encoding_dim, activation="tanh",
                activity_regularizer=regularizers
.l1(10e-5))(input_layer)
encoder = Dense(int(encoding_dim / 2), activation
="relu")(encoder)

decoder = Dense(int(encoding_dim / 2), activation
='tanh')(encoder)
decoder = Dense(input_dim, activation='relu')(dec
oder)

autoencoder = Model(inputs=input_layer, outputs=d
ecoder)
```

```
WARNING:tensorflow:From C:\Users\kuna\AppData\Loca
l\Continuum\anaconda3\lib\site-packages\tensorflow
\python\framework\op_def_library.py:263: colocate_
with (from tensorflow.python.framework.ops) is dep
recated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

In [117]:

```
nb_epoch = 100
batch_size = 32

autoencoder.compile(optimizer='adam',
                    loss='mean_squared_error',
                    metrics=['accuracy'])

checkpointer = ModelCheckpoint(filepath="model.h
5",
                               verbose=0,
                               save_best_only=Tru
e)
tensorboard = TensorBoard(log_dir='./logs',
                          histogram_freq=0,
                          write_graph=True,
                          write_images=True)

history = autoencoder.fit(X_train, X_train,
                    epochs=nb_epoch,
                    batch_size=batch_size,
                    shuffle=True,
                    validation_data=(X_test, X_te
st),
                    verbose=1,
                    callbacks=[checkpointer, tens
orboard]).history
```

```
WARNING:tensorflow:From C:\Users\kuna\AppData\Loca
```

```
l\Continuum\anaconda3\lib\site-packages\tensorflow
\python\ops\math_ops.py:3066: to_int32 (from tenso
rflow.python.ops.math_ops) is deprecated and will
be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 662 samples, validate on 268 samples
Epoch 1/100
662/662 [==============================] - 0s 596u
s/step - loss: 1747.8939 - acc: 0.3202 - val_loss:
1420.5329 - val_acc: 0.3246
Epoch 2/100
662/662 [==============================] - 0s 96u
s/step - loss: 1715.2627 - acc: 0.4577 - val_loss:
1397.2241 - val_acc: 0.3806
Epoch 3/100
662/662 [==============================] - 0s 94u
s/step - loss: 1694.7902 - acc: 0.5317 - val_loss:
1379.4577 - val_acc: 0.4627
Epoch 4/100
662/662 [==============================] - 0s 136u
s/step - loss: 1677.8870 - acc: 0.5710 - val_loss:
1363.1220 - val_acc: 0.5187
Epoch 5/100
662/662 [==============================] - 0s 127u
s/step - loss: 1662.1773 - acc: 0.6465 - val_loss:
1349.5229 - val_acc: 0.6119
Epoch 6/100
662/662 [==============================] - 0s 121u
s/step - loss: 1648.2503 - acc: 0.7462 - val_loss:
1336.2131 - val_acc: 0.6828
Epoch 7/100
662/662 [==============================] - 0s 150u
s/step - loss: 1634.8200 - acc: 0.7825 - val_loss:
1324.8954 - val_acc: 0.6791
Epoch 8/100
662/662 [==============================] - 0s 139u
s/step - loss: 1622.3483 - acc: 0.7840 - val_loss:
1313.9904 - val_acc: 0.6978
Epoch 9/100
662/662 [==============================] - 0s 133u
s/step - loss: 1610.5382 - acc: 0.7961 - val_loss:
1303.3882 - val_acc: 0.6978
Epoch 10/100
662/662 [==============================] - 0s 161u
s/step - loss: 1598.8499 - acc: 0.7795 - val_loss:
1294.1570 - val_acc: 0.7201
Epoch 11/100
662/662 [==============================] - 0s 144u
s/step - loss: 1587.5839 - acc: 0.7870 - val_loss:
1284.8723 - val_acc: 0.7127
Epoch 12/100
662/662 [==============================] - 0s 127u
s/step - loss: 1577.0516 - acc: 0.7915 - val_loss:
1275.5761 - val_acc: 0.7351
Epoch 13/100
662/662 [==============================] - 0s 119u
s/step - loss: 1566.4190 - acc: 0.7885 - val_loss:
1266.2902 - val_acc: 0.7015
Epoch 14/100
662/662 [==============================] - 0s 127u
```

```
s/step - loss: 1555.7675 - acc: 0.7825 - val_loss:
1258.4182 - val_acc: 0.7164
Epoch 15/100
662/662 [==============================] - 0s 136u
s/step - loss: 1546.1561 - acc: 0.7900 - val_loss:
1250.4131 - val_acc: 0.7201
Epoch 16/100
662/662 [==============================] - 0s 140u
s/step - loss: 1535.7799 - acc: 0.7840 - val_loss:
1241.3757 - val_acc: 0.7276
Epoch 17/100
662/662 [==============================] - 0s 130u
s/step - loss: 1526.5368 - acc: 0.7931 - val_loss:
1233.3382 - val_acc: 0.7313
Epoch 18/100
662/662 [==============================] - 0s 136u
s/step - loss: 1517.1728 - acc: 0.7870 - val_loss:
1225.9469 - val_acc: 0.7351
Epoch 19/100
662/662 [==============================] - 0s 161u
s/step - loss: 1508.1060 - acc: 0.7840 - val_loss:
1218.5700 - val_acc: 0.7500
Epoch 20/100
662/662 [==============================] - 0s 168u
s/step - loss: 1499.3984 - acc: 0.7900 - val_loss:
1210.8023 - val_acc: 0.7351
Epoch 21/100
662/662 [==============================] - 0s 150u
s/step - loss: 1489.9909 - acc: 0.7946 - val_loss:
1204.1198 - val_acc: 0.7313
Epoch 22/100
662/662 [==============================] - 0s 165u
s/step - loss: 1481.4125 - acc: 0.7915 - val_loss:
1197.8883 - val_acc: 0.7239
Epoch 23/100
662/662 [==============================] - 0s 168u
s/step - loss: 1472.7345 - acc: 0.7870 - val_loss:
1190.9940 - val_acc: 0.7351
Epoch 24/100
662/662 [==============================] - 0s 170u
s/step - loss: 1464.7214 - acc: 0.7915 - val_loss:
1185.1104 - val_acc: 0.7500
Epoch 25/100
662/662 [==============================] - 0s 167u
s/step - loss: 1456.8536 - acc: 0.7961 - val_loss:
1178.1806 - val_acc: 0.7351
Epoch 26/100
662/662 [==============================] - 0s 167u
s/step - loss: 1448.2961 - acc: 0.8021 - val_loss:
1171.8919 - val_acc: 0.7425
Epoch 27/100
662/662 [==============================] - 0s 139u
s/step - loss: 1440.6810 - acc: 0.8051 - val_loss:
1167.5713 - val_acc: 0.7575
Epoch 28/100
662/662 [==============================] - 0s 130u
s/step - loss: 1432.7027 - acc: 0.8127 - val_loss:
1162.3112 - val_acc: 0.7537
Epoch 29/100
662/662 [==============================] - 0s 131u
s/step - loss: 1425.0881 - acc: 0.8278 - val_loss:
1155.5788 - val_acc: 0.7575
```

```
Epoch 30/100
662/662 [==============================] - 0s 143u
s/step - loss: 1417.5828 - acc: 0.8233 - val_loss:
1150.9050 - val_acc: 0.7575
Epoch 31/100
662/662 [==============================] - 0s 125u
s/step - loss: 1410.3577 - acc: 0.8278 - val_loss:
1144.9596 - val_acc: 0.7649
Epoch 32/100
662/662 [==============================] - 0s 144u
s/step - loss: 1403.0831 - acc: 0.8233 - val_loss:
1139.3417 - val_acc: 0.7575
Epoch 33/100
662/662 [==============================] - 0s 125u
s/step - loss: 1396.3589 - acc: 0.8278 - val_loss:
1134.0541 - val_acc: 0.7575
Epoch 34/100
662/662 [==============================] - 0s 137u
s/step - loss: 1389.1216 - acc: 0.8187 - val_loss:
1125.7750 - val_acc: 0.7612
Epoch 35/100
662/662 [==============================] - 0s 130u
s/step - loss: 1382.3776 - acc: 0.8218 - val_loss:
1122.2548 - val_acc: 0.7649
Epoch 36/100
662/662 [==============================] - 0s 139u
s/step - loss: 1375.9275 - acc: 0.8293 - val_loss:
1116.7689 - val_acc: 0.7687
Epoch 37/100
662/662 [==============================] - 0s 128u
s/step - loss: 1369.1792 - acc: 0.8233 - val_loss:
1110.2251 - val_acc: 0.7687
Epoch 38/100
662/662 [==============================] - 0s 152u
s/step - loss: 1362.9281 - acc: 0.8293 - val_loss:
1104.8319 - val_acc: 0.7724
Epoch 39/100
662/662 [==============================] - 0s 147u
s/step - loss: 1357.4527 - acc: 0.8369 - val_loss:
1095.4479 - val_acc: 0.7612
Epoch 40/100
662/662 [==============================] - 0s 122u
s/step - loss: 1351.1118 - acc: 0.8248 - val_loss:
1097.8448 - val_acc: 0.7724
Epoch 41/100
662/662 [==============================] - 0s 124u
s/step - loss: 1345.5300 - acc: 0.8233 - val_loss:
1091.7693 - val_acc: 0.7799
Epoch 42/100
662/662 [==============================] - 0s 137u
s/step - loss: 1340.6004 - acc: 0.8278 - val_loss:
1084.5521 - val_acc: 0.7388
Epoch 43/100
662/662 [==============================] - 0s 136u
s/step - loss: 1333.4843 - acc: 0.8127 - val_loss:
1086.0786 - val_acc: 0.7873
Epoch 44/100
662/662 [==============================] - 0s 136u
s/step - loss: 1328.8553 - acc: 0.8399 - val_loss:
1077.4907 - val_acc: 0.7836
Epoch 45/100
```

```
662/662 [==============================] - 0s 130u
s/step - loss: 1322.2272 - acc: 0.8263 - val_loss:
1077.2940 - val_acc: 0.7687
Epoch 46/100
662/662 [==============================] - 0s 140u
s/step - loss: 1316.5732 - acc: 0.8233 - val_loss:
1075.3950 - val_acc: 0.7724
Epoch 47/100
662/662 [==============================] - 0s 136u
s/step - loss: 1312.1577 - acc: 0.8248 - val_loss:
1071.2736 - val_acc: 0.7985
Epoch 48/100
662/662 [==============================] - 0s 127u
s/step - loss: 1306.1514 - acc: 0.8278 - val_loss:
1066.7118 - val_acc: 0.7649
Epoch 49/100
662/662 [==============================] - 0s 139u
s/step - loss: 1300.1659 - acc: 0.8278 - val_loss:
1067.5158 - val_acc: 0.7761
Epoch 50/100
662/662 [==============================] - 0s 155u
s/step - loss: 1295.0057 - acc: 0.8384 - val_loss:
1074.2899 - val_acc: 0.7761
Epoch 51/100
662/662 [==============================] - 0s 142u
s/step - loss: 1289.0776 - acc: 0.8338 - val_loss:
1057.6597 - val_acc: 0.7985
Epoch 52/100
662/662 [==============================] - 0s 137u
s/step - loss: 1283.9296 - acc: 0.8338 - val_loss:
1065.2792 - val_acc: 0.7799
Epoch 53/100
662/662 [==============================] - 0s 127u
s/step - loss: 1277.8075 - acc: 0.8399 - val_loss:
1059.3218 - val_acc: 0.7799
Epoch 54/100
662/662 [==============================] - 0s 159u
s/step - loss: 1272.4223 - acc: 0.8308 - val_loss:
1051.8586 - val_acc: 0.7873
Epoch 55/100
662/662 [==============================] - 0s 133u
s/step - loss: 1266.9530 - acc: 0.8353 - val_loss:
1046.6197 - val_acc: 0.7761
Epoch 56/100
662/662 [==============================] - 0s 153u
s/step - loss: 1262.4101 - acc: 0.8399 - val_loss:
1040.5929 - val_acc: 0.7948
Epoch 57/100
662/662 [==============================] - 0s 150u
s/step - loss: 1256.8624 - acc: 0.8414 - val_loss:
1044.4757 - val_acc: 0.7761
Epoch 58/100
662/662 [==============================] - 0s 153u
s/step - loss: 1251.9315 - acc: 0.8444 - val_loss:
1036.1945 - val_acc: 0.7799
Epoch 59/100
662/662 [==============================] - 0s 131u
s/step - loss: 1246.9635 - acc: 0.8414 - val_loss:
1032.9058 - val_acc: 0.7985
Epoch 60/100
662/662 [==============================] - 0s 112u
s/step - loss: 1241.8630 - acc: 0.8429 - val_loss:
```

```
1031.3896 - val_acc: 0.7873
Epoch 61/100
662/662 [==============================] - 0s 139u
s/step - loss: 1236.8771 - acc: 0.8459 - val_loss:
1018.4703 - val_acc: 0.7761
Epoch 62/100
662/662 [==============================] - 0s 127u
s/step - loss: 1232.0100 - acc: 0.8429 - val_loss:
1017.2015 - val_acc: 0.7985
Epoch 63/100
662/662 [==============================] - 0s 127u
s/step - loss: 1228.2258 - acc: 0.8459 - val_loss:
1012.6642 - val_acc: 0.7575
Epoch 64/100
662/662 [==============================] - 0s 119u
s/step - loss: 1233.0131 - acc: 0.8369 - val_loss:
1041.3123 - val_acc: 0.7910
Epoch 65/100
662/662 [==============================] - 0s 137u
s/step - loss: 1226.5140 - acc: 0.8384 - val_loss:
1017.1269 - val_acc: 0.7537
Epoch 66/100
662/662 [==============================] - 0s 139u
s/step - loss: 1220.3368 - acc: 0.8278 - val_loss:
1022.0935 - val_acc: 0.7836
Epoch 67/100
662/662 [==============================] - 0s 146u
s/step - loss: 1213.7652 - acc: 0.8384 - val_loss:
1003.9931 - val_acc: 0.7799
Epoch 68/100
662/662 [==============================] - 0s 128u
s/step - loss: 1209.7167 - acc: 0.8520 - val_loss:
1009.1202 - val_acc: 0.7948
Epoch 69/100
662/662 [==============================] - 0s 143u
s/step - loss: 1204.9570 - acc: 0.8535 - val_loss:
991.0382 - val_acc: 0.7463
Epoch 70/100
662/662 [==============================] - 0s 134u
s/step - loss: 1199.1336 - acc: 0.8414 - val_loss:
994.9906 - val_acc: 0.7761
Epoch 71/100
662/662 [==============================] - 0s 136u
s/step - loss: 1194.3875 - acc: 0.8308 - val_loss:
997.5357 - val_acc: 0.7649
Epoch 72/100
662/662 [==============================] - 0s 121u
s/step - loss: 1192.0899 - acc: 0.8550 - val_loss:
992.0940 - val_acc: 0.7687
Epoch 73/100
662/662 [==============================] - 0s 122u
s/step - loss: 1187.6104 - acc: 0.8474 - val_loss:
996.0609 - val_acc: 0.7649
Epoch 74/100
662/662 [==============================] - 0s 128u
s/step - loss: 1181.8495 - acc: 0.8474 - val_loss:
991.1104 - val_acc: 0.7687
Epoch 75/100
662/662 [==============================] - 0s 144u
s/step - loss: 1176.8851 - acc: 0.8429 - val_loss:
993.5286 - val_acc: 0.7761
```

```
Epoch 76/100
662/662 [==============================] - 0s 128u
s/step - loss: 1172.3352 - acc: 0.8489 - val_loss:
991.1881 - val_acc: 0.7724
Epoch 77/100
662/662 [==============================] - 0s 170u
s/step - loss: 1167.3623 - acc: 0.8459 - val_loss:
992.2330 - val_acc: 0.7724
Epoch 78/100
662/662 [==============================] - 0s 133u
s/step - loss: 1163.3909 - acc: 0.8489 - val_loss:
988.5997 - val_acc: 0.7724
Epoch 79/100
662/662 [==============================] - 0s 150u
s/step - loss: 1159.4874 - acc: 0.8444 - val_loss:
990.5476 - val_acc: 0.7761
Epoch 80/100
662/662 [==============================] - 0s 137u
s/step - loss: 1156.1944 - acc: 0.8489 - val_loss:
995.4526 - val_acc: 0.7836
Epoch 81/100
662/662 [==============================] - 0s 139u
s/step - loss: 1152.8911 - acc: 0.8444 - val_loss:
991.1230 - val_acc: 0.7836
Epoch 82/100
662/662 [==============================] - 0s 114u
s/step - loss: 1149.7347 - acc: 0.8535 - val_loss:
989.6885 - val_acc: 0.7799
Epoch 83/100
662/662 [==============================] - 0s 136u
s/step - loss: 1152.4648 - acc: 0.8278 - val_loss:
979.2618 - val_acc: 0.7836
Epoch 84/100
662/662 [==============================] - 0s 134u
s/step - loss: 1143.6389 - acc: 0.8520 - val_loss:
966.5316 - val_acc: 0.7687
Epoch 85/100
662/662 [==============================] - 0s 140u
s/step - loss: 1138.7732 - acc: 0.8399 - val_loss:
972.5461 - val_acc: 0.7724
Epoch 86/100
662/662 [==============================] - 0s 161u
s/step - loss: 1134.1762 - acc: 0.8429 - val_loss:
959.1799 - val_acc: 0.7761
Epoch 87/100
662/662 [==============================] - 0s 125u
s/step - loss: 1130.4920 - acc: 0.8459 - val_loss:
965.1720 - val_acc: 0.7799
Epoch 88/100
662/662 [==============================] - 0s 147u
s/step - loss: 1126.0093 - acc: 0.8414 - val_loss:
963.1853 - val_acc: 0.7761
Epoch 89/100
662/662 [==============================] - 0s 142u
s/step - loss: 1124.1854 - acc: 0.8550 - val_loss:
961.3616 - val_acc: 0.7799
Epoch 90/100
662/662 [==============================] - 0s 142u
s/step - loss: 1120.4619 - acc: 0.8414 - val_loss:
960.8328 - val_acc: 0.7985
Epoch 91/100
662/662 [==============================] - 0s 153u
```

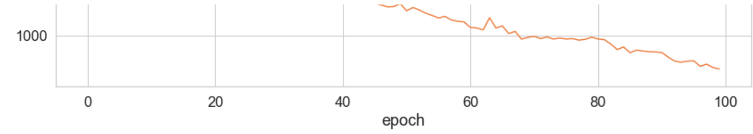
```
s/step - loss: 1116.0949 - acc: 0.8444 - val_loss:
959.5213 - val_acc: 0.7985
Epoch 92/100
662/662 [==============================] - 0s 131u
s/step - loss: 1113.0593 - acc: 0.8520 - val_loss:
948.6291 - val_acc: 0.7873
Epoch 93/100
662/662 [==============================] - 0s 146u
s/step - loss: 1108.7436 - acc: 0.8505 - val_loss:
939.5209 - val_acc: 0.8060
Epoch 94/100
662/662 [==============================] - 0s 134u
s/step - loss: 1104.7653 - acc: 0.8459 - val_loss:
936.6389 - val_acc: 0.7948
Epoch 95/100
662/662 [==============================] - 0s 147u
s/step - loss: 1102.2551 - acc: 0.8489 - val_loss:
939.3595 - val_acc: 0.7985
Epoch 96/100
662/662 [==============================] - 0s 142u
s/step - loss: 1101.4959 - acc: 0.8369 - val_loss:
940.0527 - val_acc: 0.7948
Epoch 97/100
662/662 [==============================] - 0s 137u
s/step - loss: 1097.2288 - acc: 0.8580 - val_loss:
927.0971 - val_acc: 0.7537
Epoch 98/100
662/662 [==============================] - 0s 139u
s/step - loss: 1092.3602 - acc: 0.8429 - val_loss:
932.1917 - val_acc: 0.7985
Epoch 99/100
662/662 [==============================] - 0s 127u
s/step - loss: 1094.4987 - acc: 0.8308 - val_loss:
924.5593 - val_acc: 0.7649
Epoch 100/100
662/662 [==============================] - 0s 152u
s/step - loss: 1088.4686 - acc: 0.8520 - val_loss:
920.7289 - val_acc: 0.7724
```

In [118]:

```
# Evaluation
```

In [119]:

```
plt.plot(history['loss'])
plt.plot(history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right');
```



The reconstruction error on our training and test data seems to converge nicely. Is it low enough? Let's have a closer look at the error distribution:

In [120]:

```
predictions = autoencoder.predict(X_test)
```

In [121]:

```
mse = np.mean(np.power(X_test - predictions, 2), axis=1)
error_df = pd.DataFrame({'reconstruction_error': mse,
                        'true_class': y_test})
```

In [122]:

```
error_df.describe()
```

Out[122]:

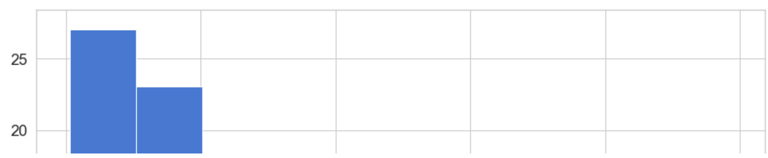
	reconstruction_error	true_class
count	268.000000	268.000000
mean	919.749266	0.361940
std	3652.005677	0.481461
min	0.049912	0.000000
25%	2.512415	0.000000
50%	15.145470	0.000000
75%	408.687194	1.000000
max	39546.021756	1.000000

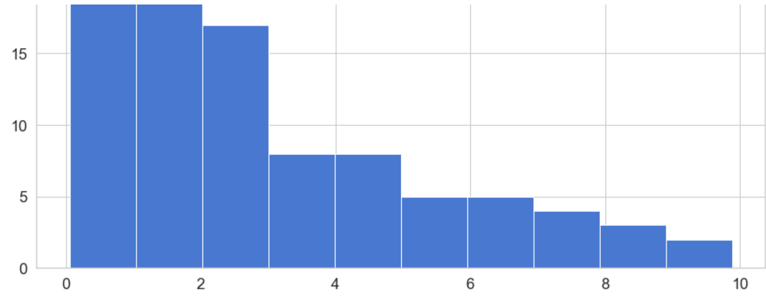
In [123]:

```
# Reconstruction error without Attack
```

In [124]:

```
fig = plt.figure()
ax = fig.add_subplot(111)
normal_error_df = error_df[(error_df['true_class']== 0) & (error_df['reconstruction_error'] < 10)]
_ = ax.hist(normal_error_df.reconstruction_error.values, bins=10)
```

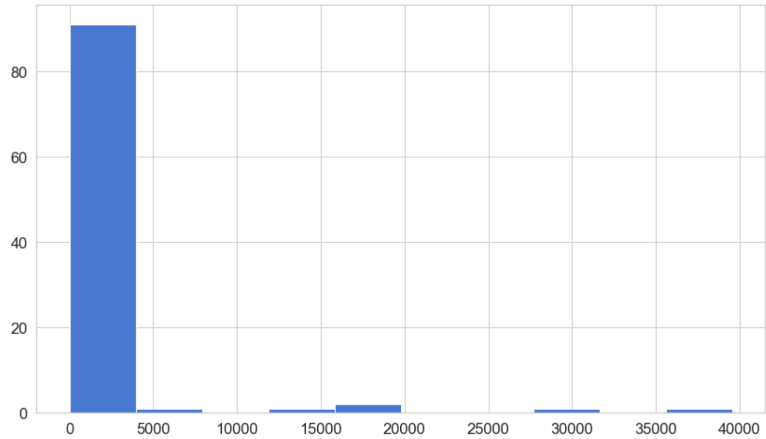




Reconstruction error with Attack

In [125]:

```
fig = plt.figure()
ax = fig.add_subplot(111)
fraud_error_df = error_df[error_df['true_class']
== 1]
_ = ax.hist(fraud_error_df.reconstruction_error.v
alues, bins=10)
```



In [126]:

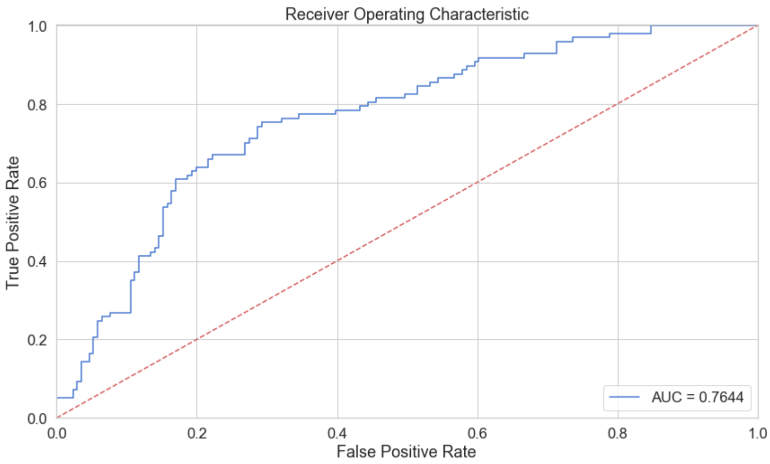
```
from sklearn.metrics import (confusion_matrix, pr
ecision_recall_curve, auc,
                             roc_curve, recall_sc
ore, classification_report, f1_score,
                             precision_recall_fsc
ore_support)
```

ROC curves are very useful tool for understanding the performance of binary classifiers. However, our case is a bit out of the ordinary. We have a very imbalanced dataset. Nonetheless, let's have a look at our ROC curve:

In [127]:

```
fpr, tpr, thresholds = roc_curve(error_df.true_cl
ass, error_df.reconstruction_error)
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, label='AUC = %0.4f'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
```

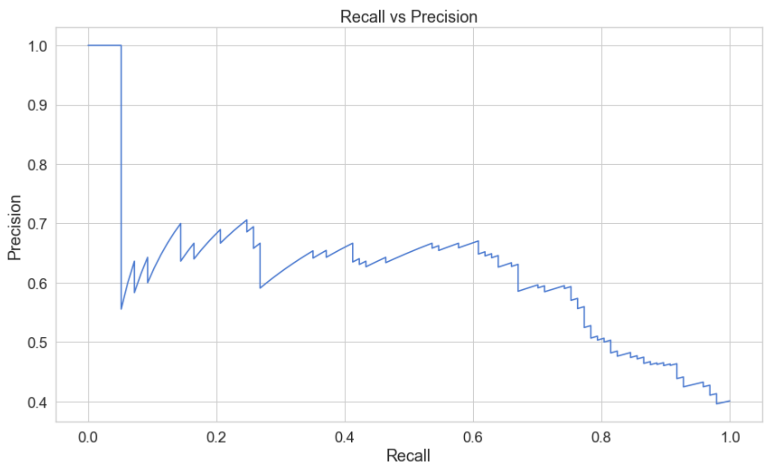
```
plt.xlim([-0.001, 1])
plt.ylim([0, 1.001])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show();
```



The ROC curve plots the true positive rate versus the false positive rate, over different threshold values. Basically, we want the blue line to be as close as possible to the upper left corner. While our results look pretty good, we have to keep in mind of the nature of our dataset. ROC doesn't look very useful for us. Onward...

In [128]:

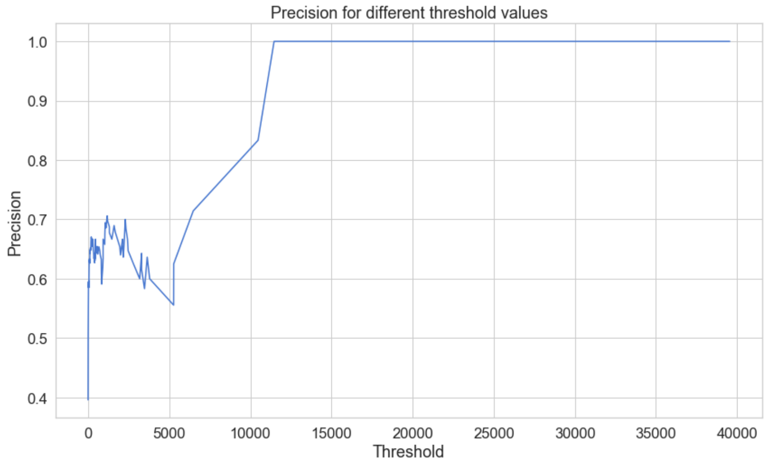
```
precision, recall, th = precision_recall_curve(error_df.true_class, error_df.reconstruction_error)
plt.plot(recall, precision, 'b', label='Precision-Recall curve')
plt.title('Recall vs Precision')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()
```



A high area under the curve represents both high recall and high precision, where high precision relates to a low false positive rate, and high recall relates to a low false negative rate. High scores for both show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).

In [129]:

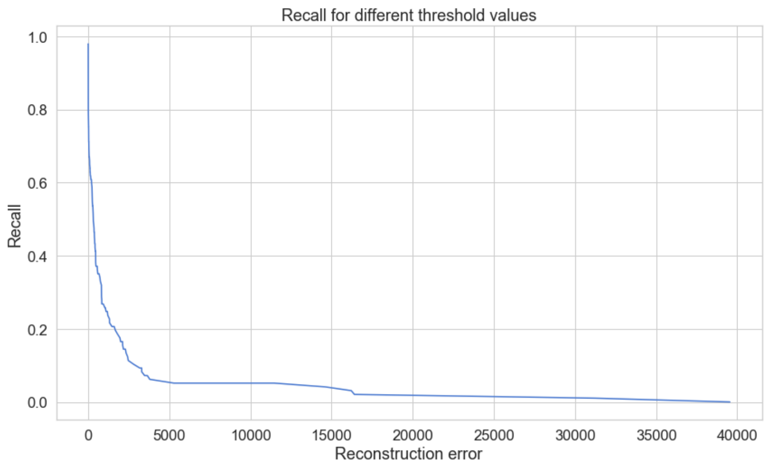
```
plt.plot(th, precision[1:], 'b', label='Threshold
-Precision curve')
plt.title('Precision for different threshold valu
es')
plt.xlabel('Threshold')
plt.ylabel('Precision')
plt.show()
```



You can see that as the reconstruction error increases our precision rises as well. Let's have a look at the recall:

In [130]:

```
plt.plot(th, recall[1:], 'b', label='Threshold-Re
call curve')
plt.title('Recall for different threshold values'
)
plt.xlabel('Reconstruction error')
plt.ylabel('Recall')
plt.show()
```



Here, we have the exact opposite situation. As the reconstruction error increases the recall decreases.

Prediction Our model is a bit different this time. It doesn't know how to predict new values. But we don't need that. In order to predict whether or not a new/unseen system call sequence is normal or attack, we'll calculate the reconstruction error from the systemcall data itself. If the error is larger than a predefined threshold, we'll mark it as a attack (since our model should have a low error on normal transactions). Let's pick that value:

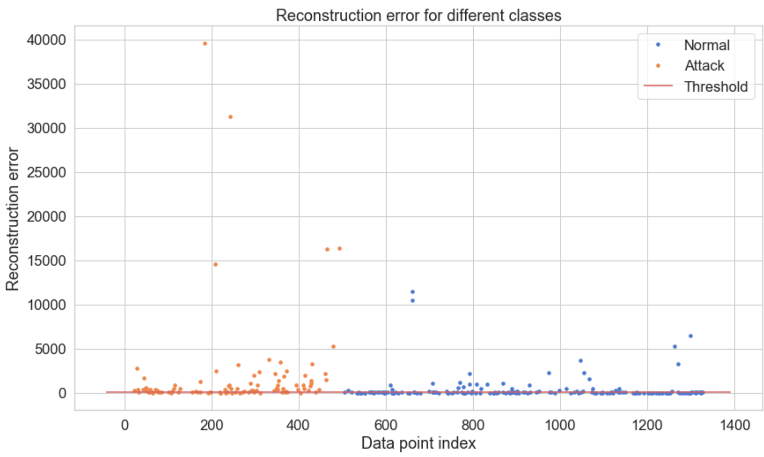
In [139]:

```
threshold = 20
```

And see how well we're dividing the two types of transactions:

In [140]:

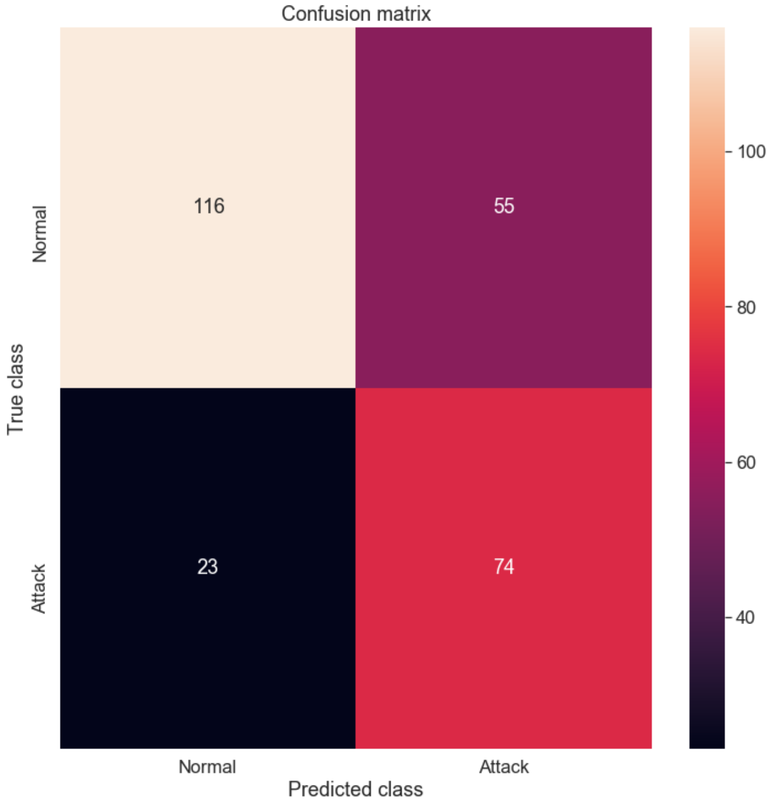
```
groups = error_df.groupby('true_class')
fig, ax = plt.subplots()

for name, group in groups:
    ax.plot(group.index, group.reconstruction_err
or, marker='o', ms=3.5, linestyle='',
            label= "Attack" if name == 1 else "No
rmal")
ax.hlines(threshold, ax.get_xlim()[0], ax.get_xli
m()[1], colors="r", zorder=100, label='Threshold'
)
ax.legend()
plt.title("Reconstruction error for different cla
sses")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show();
```



I know, that chart might be a bit deceiving. Let's have a look at the confusion matrix:

In [141]:

```
y_pred = [1 if e > threshold else 0 for e in erro
r_df.reconstruction_error.values]
conf_matrix = confusion_matrix(error_df.true_clas
s, y_pred)

plt.figure(figsize=(12, 12))
sns.heatmap(conf_matrix, xticklabels=LABELS, ytic
klabels=LABELS, annot=True, fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```



Our model seems to catch a lot of the attack cases. Of course, there is a catch (see what I did there?). The number of normal transactions classified as Attack is really high. Is this really a problem? Probably it is. You might want to increase or decrease the value of the threshold, depending on the problem. That one is up to you.

Conclusion

We've created a very simple Deep Autoencoder in Keras that can reconstruct what normal system call looks like. Initially, I was a bit skeptical about whether or not this whole thing is gonna work out, bit it kinda did. Think about it, we gave a lot of one-class examples (normal systemcalls) to a model and it learned (somewhat) how to discriminate whether or not new examples belong to that same class. Isn't that cool? Our dataset was kind of magical, though. We really don't know what the original features look like.

Keras gave us very clean and easy to use API to build a non-trivial Deep Autoencoder. You can search for TensorFlow implementations and see for yourself how much boilerplate you need in order to train one. Can you apply a similar model to a different problem?

References

Building Autoencoders in Keras Stanford tutorial on Autoencoders Stacked Autoencoders in TensorFlow

LSTM Autoencoder Classifier Model

In [185]:

```
# LsTM autoencoder - https://machinelearningmaste
ry.com/lstm-autoencoders/
# LSTM Autoencoder - https://towardsdatascience.c
om/lstm-autoencoder-for-extreme-rare-event-classi
fication-in-keras-ce209a224cfb
```

Prepare data for LSTM models

LSTM is a bit more demanding than other models. Significant amount of time and attention goes in preparing the data that fits an LSTM.

First, we will create the 3-dimensional arrays of shape: (samples x timesteps x features). Samples mean the number of data points. Timesteps is the number of time steps we look back at any time t to make a prediction. This is also referred to as lookback period. The features is the number of features the data has, in other words, the number of predictors in a multivariate data.

In [142]:

```
result.head()
```

Out[142]:

	Label	168	265	3	54	162	142	309	146	114	175
0	1	193	75	0	0	0	0	0	0	0	0
1	1	0	110	139	0	0	286	0	55	0	64
2	1	249	133	112	0	0	0	0	0	0	0
3	1	0	1	51	809	0	0	202	0	0	0
4	1	426	234	157	0	0	0	0	0	0	0

In [148]:

```
input_X = result.loc[:, result.columns != 'Label'
].values  # converts the df to a numpy array
input_y = result['Label'].values

n_features = input_X.shape[1]  # number of featur
es
```

In [144]:

```
def temporalize(X, y, lookback):
    output_X = []
    output_y = []
    for i in range(len(X)-lookback-1):
        t = []
```

```
        t = []
        for j in range(1,lookback+1):
            # Gather past records upto the lookba
ck period
            t.append(X[[(i+j+1)], :])
        output_X.append(t)
        output_y.append(y[i+lookback+1])
    return output_X, output_y
```

In LSTM, to make prediction at any time t, we will look at data from (t-lookback):t. In the following, we have an example to show how the input data are transformed with the temporalize function with lookback=5. For the modeling, we may use a longer lookback.

In [146]:

```
'''
Test: The 3D tensors (arrays) for LSTM are formin
g correctly.
'''
print('First instance of y = 1 in the original da
ta')
display(result.iloc[(np.where(np.array(input_y) =
= 1)[0][0]-5):(np.where(np.array(input_y) == 1)[0
][0]+1), ])

lookback = 5  # Equivalent to 10 min of past dat
a.
# Temporalize the data
X, y = temporalize(X = input_X, y = input_y, look
back = lookback)

print('For the same instance of y = 1, we are kee
ping past 5 samples in the 3D predictor array,
 X.')
display(pd.DataFrame(np.concatenate(X[np.where(np
.array(y) == 1)[0][0]], axis=0 )))
```

First instance of y = 1 in the original data

	Label	168	265	3	54	162	142	309	146	114	175	43

For the same instance of y = 1, we are keeping past 5 samples in the 3D predictor array, X.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	249	133	112	0	0	0	0	0	0	0	60	0	0	0
1	0	1	51	809	0	0	202	0	0	0	0	0	0	0
2	426	234	157	0	0	0	0	0	0	0	0	0	0	2
3	227	115	90	1	3	0	0	40	0	0	0	0	0	10
4	0	0	0	0	325	0	0	0	98	0	0	0	0	0

Divide the data into train, valid, and test

In [177]:

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns

import pandas as pd
import numpy as np
from pylab import rcParams

import tensorflow as tf
from keras import optimizers, Sequential
from keras.models import Model
from keras.utils import plot_model
from keras.layers import Dense, LSTM, RepeatVecto
r, TimeDistributed
from keras.callbacks import ModelCheckpoint, Tens
orBoard

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_sp
lit
from sklearn.metrics import confusion_matrix, pre
cision_recall_curve
from sklearn.metrics import recall_score, classif
ication_report, auc, roc_curve
from sklearn.metrics import precision_recall_fsco
re_support, f1_score

from numpy.random import seed
seed(7)
from tensorflow import set_random_seed
set_random_seed(11)

from sklearn.model_selection import train_test_sp
lit

SEED = 123 #used to help randomly select the data
points
DATA_SPLIT_PCT = 0.2

rcParams['figure.figsize'] = 8, 6
LABELS = ["Normal","Attack"]

X_train, X_test, y_train, y_test = train_test_spl
it(np.array(X), np.array(y), test_size=DATA_SPLIT
_PCT, random_state=SEED)
X_train, X_valid, y_train, y_valid = train_test_s
plit(X_train, y_train, test_size=DATA_SPLIT_PCT,
random_state=SEED)
```

In [151]:

```
X_train.shape
```

Out[151]:

```
(852, 5, 1, 87)
```

In [152]:

```
X_train_y0 = X_train[y_train==0]
X_train_y1 = X_train[y_train==1]

X_valid_y0 = X_valid[y_valid==0]
X_valid_y1 = X_valid[y_valid==1]
```

In [153]:

```
X_train_y0.shape
```

Out[153]:

```
(542, 5, 1, 87)
```

Reshaping the data

The tensors we have here are 4-dimensional. We will reshape them into the desired 3-dimensions corresponding to sample x lookback x features.

In [154]:

```
X_train = X_train.reshape(X_train.shape[0], lookb
ack, n_features)
X_train_y0 = X_train_y0.reshape(X_train_y0.shape[
0], lookback, n_features)
X_train_y1 = X_train_y1.reshape(X_train_y1.shape[
0], lookback, n_features)

X_test = X_test.reshape(X_test.shape[0], lookback
, n_features)

X_valid = X_valid.reshape(X_valid.shape[0], lookb
ack, n_features)
X_valid_y0 = X_valid_y0.reshape(X_valid_y0.shape[
0], lookback, n_features)
X_valid_y1 = X_valid_y1.reshape(X_valid_y1.shape[
0], lookback, n_features)
```

In [155]:

```
n_features
```

Out[155]:

```
87
```

Standardize the data It is usually better to use a standardized data (transformed to Gaussian, mean 0 and sd 1) for autoencoders.

One common mistake is: we normalize the entire data and then split into train-test. This is not correct. Test data should be completely unseen to anything during the modeling. We should normalize the test data using the feature summary statistics computed from the training data. For normalization, these statistics are the mean and variance for each feature.

The same logic should be used for the validation set. This

The same logic should be used for the validation set. This makes the model more stable for a test data.

To do this, we will require two UDFs.

flatten: This function will re-create the original 2D array from which the 3D arrays were created. This function is the inverse of temporalize, meaning X = flatten(temporalize(X)). scale: This function will scale a 3D array that we created as inputs to the LSTM.

In [156]:

```
def flatten(X):
    '''
    Flatten a 3D array.

    Input
    X            A 3D array for lstm, where the a
rray is sample x timesteps x features.

    Output
    flattened_X  A 2D array, sample x features.
    '''
    flattened_X = np.empty((X.shape[0], X.shape[2
]))  # sample x features array.
    for i in range(X.shape[0]):
        flattened_X[i] = X[i, (X.shape[1]-1), :]
    return(flattened_X)

def scale(X, scaler):
    '''
    Scale 3D array.
```