First attempt at a model - Group 4 (Mayuresh Mali, Divya G Tripathi, Devika Antarkar, Soma Ghosh, Kshitij Pathak)

Import Libraries

In [1]:
```
import sklearn.feature_extraction
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
import tldextract
import matplotlib as plt
from pylab import *
import re
import math
from itertools import groupby
```

Read the data

In [2]:
```
train_data = pd.read_csv('train_data.csv')
train_data_copy = train_data
test_data = pd.read_csv('test_data.csv')
```

In [3]:
```
import tldextract

def domain_extract(uri):
    ext = tldextract.extract(uri)
    if (not ext.suffix):
        return np.nan
    else:
        return ext.domain

train_data_copy['domain'] = [ domain_extract(DNS_Address) for DNS_Address in train_data_copy['DNS_Address']]
del train_data_copy['DNS_Address']
train_data_copy.head()
```

Out[3]:

	DNS_Type	domain
0	benign	dr-beckmann
1	benign	lifeadvicedaily
2	benign	smiles
3	benign	axenprovencetourism
4	benign	ayto-alcaladehenares

In [4]:
```
train_data_copy['domain'] = train_data_copy['domain'].astype(str)
```

Feature Engineering- Adding few more features

Let me create some features which can be useful such as length of domain name, number of vowels in it, consecutive consonants and digits in the domain name and also since XGBoost needs numeric features, we need to encode the data as numeric.

In [5]:
```
# Add a length field for the domain
train_data_copy['length'] = [len(x) for x in train_data_copy['domain']]
```

In [6]:
```
train_data_copy.head()
```

Out[6]:

	DNS_Type	domain	length
0	benign	dr-beckmann	11
1	benign	lifeadvicedaily	15
2	benign	smiles	6
3	benign	axenprovencetourism	20
4	benign	ayto-alcaladehenares	20

In [7]:
```
# Calculating character entropy
import math
from collections import Counter

def entropy(s):
    p, lns = Counter(s), float(len(s))
    return -sum( count/lns * math.log(count/lns, 2) for count in p.values())
```

In [8]:
```
def tri_gram(domain):
    s = []
    count = 2
    while count < len(domain):
        s.append(domain[count - 2] + domain[count - 1] + domain[count])
        count = count + 1
    return s

dataset = {}
sum_of_frequency = 0.0
# load trigrams_google into dataset

with open("trigram_google.txt") as f:
    for line in f:
        (key, val) = line.split()
        dataset[key] = float(val)
        sum_of_frequency += dataset[key]


# print sum
def calc_freq(trigrams):
    frequency = sum([dataset.get(trigram, 0) for trigram in trigrams]) / sum_of_frequency
    return frequency
```

In [9]:
```
# calculate vowels
def calc_vowels(y):
    num_vowel = 0
    vowels = list('aeiou')
    for char in y:
        if char in vowels:
            num_vowel += 1

    return num_vowel

#Calculate number of digits
def calc_digits(z):
    num_digit = 0
    digits = list('0123456789')
    for char in z:
        if char in digits:
            num_digit += 1

    return num_digit
```

In [10]:
```
# Maximum length of Consecutive consonants
def consecutive_consonants(string):
    from itertools import groupby
    is_vowel = lambda char: char in "aAeEiIoOuU"
    best = 0
    listnames = ["".join(g) for v, g in groupby(string, key=is_vowel) if not v]
    for index in range(len(listnames)):
        if len(listnames[index]) > best:
            best = len(listnames[index])
    return best
```

In [11]:
```
# Add a entropy field for the domain
train_data_copy['entropy'] = [entropy(x) for x in train_data_copy['domain']]
```

In [12]:
```
# Add a trigram field for the domain
train_data_copy['trigrams'] = [tri_gram(x) for x in train_data_copy['domain']]
```

In [13]:
```
# Add a trigram frequency field for the domain
train_data_copy['trigram_freq'] = [calc_freq(x) for x in train_data_copy['domain']]
```

In [14]:
```
# Add a vowels field for the domain
train_data_copy['vowels'] = [calc_vowels(x) for x in train_data_copy['domain']]
```

In [15]:
```
# Add a digits field for the domain
train_data_copy['digits'] = [calc_digits(x) for x in train_data_copy['domain']]
```

In [16]:
```
# Add a consec_consonants field for the domain
train_data_copy['consec_consonants'] = [consecutive_consonants(x) for x in train_data_copy['domain']]
```

In [17]:
```
train_data_copy['threat_type'] = train_data_copy.DNS_Type.apply(lambda x: 0 if x=='benign' else 1)
```

In [18]:
```
train_data_copy.head()
```

Out[18]:

	DNS_Type	domain	length	entropy	trigrams	trigram_freq	vowels	digits	consec_consonants	threat_type
0	benign	dr-beckmann	11	3.277613	[dr-, r-b, -be, bec, eck, ckm, kma, man, ann]	0.0	2	0	4	0
1	benign	lifeadvicedaily	15	3.056565	[lif, ife, fea, ead, adv, dvi, vic, ice, ced, ...	0.0	7	0	2	0
2	benign	smiles	6	2.251629	[smi, mil, ile, les]	0.0	2	0	2	0
3	benign	axenprovencetourism	20	3.684184	[aix, xe, xen, enp, npr, pro, rov, ove, ven, ...	0.0	9	0	3	0
4	benign	ayto-alcaladehenares	20	3.403702	[ayt, yto, to-, o-a, -al, alc, lca, cal, ala, ...	0.0	9	0	2	0

In [19]:
```
train_data_copy.tail()
```

Out[19]:

	DNS_Type	domain	length	entropy	trigrams	trigram_freq	vowels	digits	consec_consonants	threat_type
999995	dga	undercircumstancestyranny	25	3.463465	[und, nde, der, erc, rci, cir, irc, rcu, cum, ...	0.0	7	0	4	1
999996	dga	suchofconsentourabusers	22	3.390805	[suc, uch, cho, hof, ofc, fco, con, ons, nse, ...	0.0	9	0	2	1
999997	dga	emekcwlokmekylca	16	2.602217	[eme, mek, ekc, kcw, cwk, wka, kwk, wkm, kmw, ...	0.0	3	0	12	1
999998	dga	cdkokoednvlk	12	2.855389	[cdk, dko, kok, oko, ked, edn, dnv, nvl, ...	0.0	3	0	5	1
999999	dga	pboownxeeaysbtcg	15	3.506891	[pbo, boo, oow, own, wnx, nxe, xee, eea, eay, ...	0.0	5	0	5	1

In [20]:
```
#Rearranging the order of columns
train_data_copy = train_data_copy[['domain','DNS_Type','threat_type','length','entropy','trigrams','trigram_freq','vowels','digits','consec_consonants']]
```

In [21]:
```
train_data_copy.head()
```

Out[21]:

	domain	DNS_Type	threat_type	length	entropy	trigrams	trigram_freq	vowels	digits	consec_consonants
0	dr-beckmann	benign	0	11	3.277613	[dr-, r-b, -be, bec, eck, ckm, kma, man, ann]	0.0	2	0	4
1	lifeadvicedaily	benign	0	15	3.056565	[lif, ife, fea, ead, adv, dvi, vic, ice, ced, ...	0.0	7	0	2
2	smiles	benign	0	6	2.251629	[smi, mil, ile, les]	0.0	2	0	2
3	axenprovencetourism	benign	0	20	3.684184	[aix, xe, xen, enp, npr, pro, rov, ove, ven, ...	0.0	9	0	3
4	ayto-alcaladehenares	benign	0	20	3.403702	[ayt, yto, to-, o-a, -al, alc, lca, cal, ala, ...	0.0	9	0	2

In [22]:
```
# # selecting rows based on condition
# rslt_df = train_data_copy[train_data_copy['trigram_freq'] > 0]

# print('\nResult dataframe :\n', rslt_df)
```

Model attempt 1

In [25]:
```
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, f1_score

X = train_data_copy.as_matrix(['length', 'entropy', 'vowels', 'digits','consec_consonants'])

y = np.array(train_data_copy['threat_type'].tolist())

# Train on a 80/20 split
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)

model = XGBClassifier(silent=False,
                      scale_pos_weight=1,
                      learning_rate=0.1,
                      colsample_bytree = 0.4,
                      subsample = 0.8,
                      objective='binary:logistic',
                      n_estimators=100,
                      reg_alpha = 0.3,
                      max_depth=4,
                      gamma=10)
model.fit(X_train, y_train)
```

```
C:\Users\mayur\AppData\Local\Continuum\anaconda3\lib\site-packages\ipykernel_launcher.py:5: FutureWarning: Method .as_matrix will be removed in a future version. Use .values instead.
  """
```

Out[25]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.4, gamma=10,
              learning_rate=0.1, max_delta_step=0, max_depth=4,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0.3, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=False, subsample=0.8, verbosity=1)
```

In [26]:
```
##Testing the model
y_pred = model.predict(X_test)
predictions = y_pred
```

In [27]:
```
#Evaluating predictions
accuracy = accuracy_score(y_test, predictions)
f1_score = f1_score(y_test, predictions)
print("Accuracy: %.3f%%" % (accuracy * 100.0))
print("F1_score:", f1_score)
```

```
Accuracy: 88.599%
F1_score: 0.8885478273620412
```

Model 1: Predictions on test data

In [28]:
```
def test_it(domain):
    _X = [len(domain), entropy(domain), calc_vowels(domain), calc_digits(domain), consecutive_consonants(domain)]
    return _X
```

In [29]:
```
# Extracting just the second-level domain name from test data set rows.
test_data['newdomain'] = [ domain_extract(x) for x in test_data['domain']]
#del test_data['domain']
test_data.head()

# Adding the features to the test data set
testdomain = test_data.newdomain.apply(lambda x : test_it(x))
```

In [30]:
```
X2 = np.array(testdomain.tolist())
```

In [31]:
```
##Testing the model
y_pred2 = model.predict(X2)
predictions = [round(value) for value in y_pred2]
y_pred2
```

Out[31]:
```
array([0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1,
       0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1,
       0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0])
```

In [32]:
```
test_data['threat']= y_pred2
```

In [33]:
```
test_data.threat.value_counts()
```

Out[33]:
```
0    303
1    197
Name: threat, dtype: int64
```

Saving the data with predictions to a file

In [34]:
```
#Re-encode dga and benign labels
test_data['threat'] = test_data.threat.apply(lambda x: 'benign' if x==0 else 'dga')
del test_data['newdomain']
```

In [36]:
```
#Save output to file
import os
#os.remove("submission_output3.csv")
test_data.to_csv('submission_output3.csv', index=False)
```

We also tried another model with character encoding for letters a-z, numerals 0-9 and special characters to make 40 numbers denoting one for every character of the domain name. But we prefer this model as the accuracy and F1_score are slightly better in this model. We will try different ensemble methods and feature engineering techniques (such as n-gram probabilities and dictionary check or letter label encodings etc.) in future to improve accuracy of the model.

In [ ]:
```

```