**DGA botnet detection and classification**

**Introduction**:

Domain Generation Algorithms (DGAs) are techniques used by malware authors to dynamically generate a large number of domain names to connect to Command and Control (C&C) servers. These domain names are often difficult to predict and detect using traditional methods, posing a significant challenge for cybersecurity professionals. Detecting and classifying DGA-generated domains are critical for identifying and mitigating potential botnet threats.

**Objective**:

The objective of this project is to develop a machine learning model capable of detecting and classifying DGA-generated domains for botnet detection. The model should accurately distinguish between legitimate domain names and those generated by DGAs associated with botnets.

**Dataset:**

The dataset comprises a collection of domain names labeled as either legitimate or DGA-generated.

Sources for the dataset include:

Legitimate domain names from reputable sources such as the Alexa website ranking.

Known DGA-generated domain names obtained from threat intelligence feeds, malware analysis, or security research.

Each domain name is associated with its corresponding label (legitimate or DGA-generated).

Features may include domain length, character n-grams, entropy, vowel/consonant ratio, lexical features, and potentially others obtained through feature engineering.

**Code:**

**Import the Libraries**

* Pandas (<http://pandas.pydata.org/pandas-docs/stable/>)
* Numpy (<https://docs.scipy.org/doc/numpy/reference/>)
* Matplotlib (<http://matplotlib.org/api/pyplot_api.html>)
* Scikit-learn (<http://scikit-learn.org/stable/documentation.html>)
* YellowBrick (<http://www.scikit-yb.org/en/latest/>)
* Seaborn ([https://seaborn.pydata.org](https://seaborn.pydata.org/))
* Lime (<https://github.com/marcotcr/lime>)

*# Load Libraries - Make sure to run this cell!*

**import** pandas **as** pd

**import** numpy **as** np

**import** re

**from** collections **import** Counter

**from** sklearn **import** feature\_extraction, tree, model\_selection, metrics

**from** yellowbrick.classifier **import** ClassificationReport

**from** yellowbrick.classifier **import** ConfusionMatrix

**import** matplotlib.pyplot **as** plt

**import** matplotlib

**import** lime

**import** io

**%matplotlib** inline

**Load Features and Labels**

df\_final **=** pd**.**read\_csv('dga\_features\_final\_df.csv')

*#If you didn't get a working dataset, uncomment this line*

*#df\_final = pd.read\_csv('our\_data\_dga\_features\_final\_df.csv')*

print(df\_final**.**isDGA**.**value\_counts())

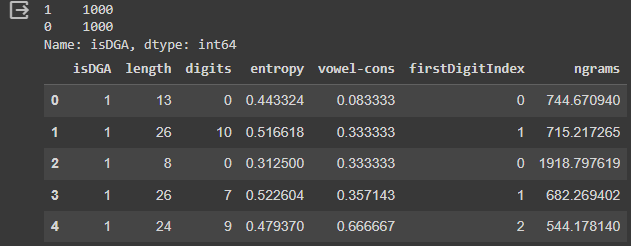
df\_final**.**head()

*# Load dictionary of common english words from part 1*

**from** six.moves **import** cPickle **as** pickle

**with** open('d\_common\_en\_words' **+** '.pickle', 'rb') **as** f:

d **=** pickle**.**load(f)

****

**Prepare Feature matrix and target vector containing the URL labels**

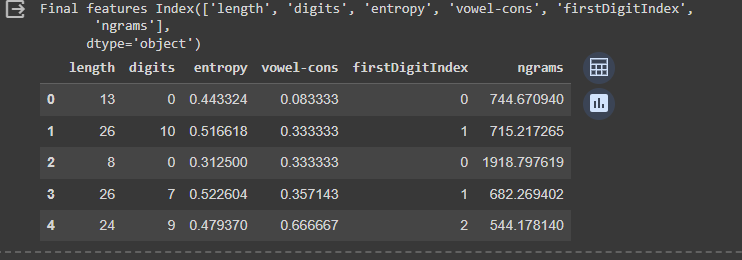
* In statistics, the feature matrix is often referred to as X
* target is a vector containing the labels for each URL (often also called y in statistics)
* In sklearn both the input and target can either be a pandas DataFrame/Series or numpy array/vector respectively (can't be lists!)
* assign 'isDGA' column to a pandas Series named 'target'
* drop 'isDGA' column from dga DataFrame and name the resulting pandas DataFrame 'feature\_matrix'

target **=** df\_final['isDGA']

feature\_matrix **=** df\_final**.**drop(['isDGA'], axis**=**1)

print('Final features', feature\_matrix**.**columns)

feature\_matrix**.**head()

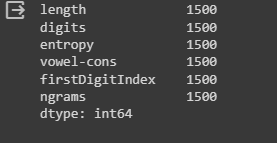


**Simple Cross-Validation**

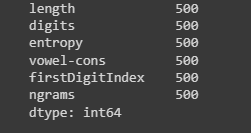
*# Simple Cross-Validation: Split the data set into training and test data*

feature\_matrix\_train, feature\_matrix\_test, target\_train, target\_test **=** model\_selection**.**train\_test\_split(feature\_matrix, target, test\_size**=**0.25, random\_state**=**33)

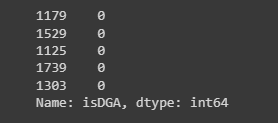
feature\_matrix\_train**.**count()



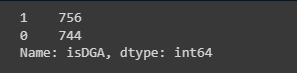
feature\_matrix\_test**.**count()



target\_train**.**head()



target\_train**.**value\_counts()



**Train the model and make a prediction**

* Using the sklearn [tree.DecisionTreeClassfier()](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html), creating a decision tree with standard parameters, and train it using the .fit() function with X\_train and target\_train data.
* Next, pull a few random rows from the data and see if your classifier got it correct.

*# Train the decision tree based on the entropy criterion*

clf **=** tree**.**DecisionTreeClassifier() *# clf means classifier*

clf **=** clf**.**fit(feature\_matrix\_train, target\_train)

*# Extract a row from the test data*

test\_feature **=** feature\_matrix\_test[192:193]

test\_target **=** target\_test[192:193]

*# Make the prediction*

pred **=** clf**.**predict(test\_feature)

print('Predicted class:', pred)

print('Accurate prediction?', pred[0] **==** test\_target)

pred[0] **==** test\_target

**Making a Prediction**

The code below demonstrates how you will go from an unknown raw domain to predicting whether it is DGA or not.

*# For simplicity let's just copy the needed function in here again*

**def** H\_entropy (x):

*# Calculate Shannon Entropy*

prob **=** [ float(x**.**count(c)) **/** len(x) **for** c **in** dict**.**fromkeys(list(x)) ]

H **=** **-** sum([ p **\*** np**.**log2(p) **for** p **in** prob ])

**return** H

**def** firstDigitIndex( s ):

**for** i, c **in** enumerate(s):

**if** c**.**isdigit():

**return** i **+** 1

**return** 0

**def** vowel\_consonant\_ratio (x):

*# Calculate vowel to consonant ratio*

x **=** x**.**lower()

vowels\_pattern **=** re**.**compile('([aeiou])')

consonants\_pattern **=** re**.**compile('([b-df-hj-np-tv-z])')

vowels **=** re**.**findall(vowels\_pattern, x)

consonants **=** re**.**findall(consonants\_pattern, x)

**try**:

ratio **=** len(vowels) **/** len(consonants)

**except**: *# catch zero devision exception*

ratio **=** 0

**return** ratio

*# ngrams: Implementation according to Schiavoni 2014: "Phoenix: DGA-based Botnet Tracking and Intelligence"*

*# http://s2lab.isg.rhul.ac.uk/papers/files/dimva2014.pdf*

**def** ngrams(word, n):

*# Extract all ngrams and return a regular Python list*

*# Input word: can be a simple string or a list of strings*

*# Input n: Can be one integer or a list of integers*

*# if you want to extract multipe ngrams and have them all in one list*

l\_ngrams **=** []

**if** isinstance(word, list):

**for** w **in** word:

**if** isinstance(n, list):

**for** curr\_n **in** n:

ngrams **=** [w[i:i**+**curr\_n] **for** i **in** range(0,len(w)**-**curr\_n**+**1)]

l\_ngrams**.**extend(ngrams)

**else**:

ngrams **=** [w[i:i**+**n] **for** i **in** range(0,len(w)**-**n**+**1)]

l\_ngrams**.**extend(ngrams)

**else**:

**if** isinstance(n, list):

**for** curr\_n **in** n:

ngrams **=** [word[i:i**+**curr\_n] **for** i **in** range(0,len(word)**-**curr\_n**+**1)]

l\_ngrams**.**extend(ngrams)

**else**:

ngrams **=** [word[i:i**+**n] **for** i **in** range(0,len(word)**-**n**+**1)]

l\_ngrams**.**extend(ngrams)

*# print(l\_ngrams)*

**return** l\_ngrams

**def** ngram\_feature(domain, d, n):

*# Input is your domain string or list of domain strings*

*# a dictionary object d that contains the count for most common english words*

*# finally you n either as int list or simple int defining the ngram length*

*# Core magic: Looks up domain ngrams in english dictionary ngrams and sums up the*

*# respective english dictionary counts for the respective domain ngram*

*# sum is normalized*

l\_ngrams **=** ngrams(domain, n)

*# print(l\_ngrams)*

count\_sum**=**0

**for** ngram **in** l\_ngrams:

**if** d[ngram]:

count\_sum**+=**d[ngram]

**try**:

feature **=** count\_sum**/**(len(domain)**-**n**+**1)

**except**:

feature **=** 0

**return** feature

**def** average\_ngram\_feature(l\_ngram\_feature):

*# input is a list of calls to ngram\_feature(domain, d, n)*

*# usually you would use various n values, like 1,2,3...*

**return** sum(l\_ngram\_feature)**/**len(l\_ngram\_feature)

In [17]:

**def** is\_dga(domain, clf, d):

*# Function that takes new domain string, trained model 'clf' as input and*

*# dictionary d of most common english words*

*# returns prediction*

domain\_features **=** np**.**empty([1,6])

*# order of features is ['length', 'digits', 'entropy', 'vowel-cons', firstDigitIndex, 'ngrams']*

domain\_features[0,0] **=** len(domain)

pattern **=** re**.**compile('([0-9])')

domain\_features[0,1] **=** len(re**.**findall(pattern, domain))

domain\_features[0,2] **=** H\_entropy(domain)

domain\_features[0,3] **=** vowel\_consonant\_ratio(domain)

domain\_features[0,4] **=** firstDigitIndex(domain)

domain\_features[0,5] **=** average\_ngram\_feature([ngram\_feature(domain, d, 1),

ngram\_feature(domain, d, 2),

ngram\_feature(domain, d, 3)])

pred **=** clf**.**predict(domain\_features)

**return** pred[0]

print('Predictions of domain %s is [0 means legit and 1 dga]: ' **%**('spardeingeld'), is\_dga('spardeingeld', clf, d))

print('Predictions of domain %s is [0 means legit and 1 dga]: ' **%**('google'), is\_dga('google', clf, d))

print('Predictions of domain %s is [0 means legit and 1 dga]: ' **%**('1vxznov16031kjxneqjk1rtofi6'), is\_dga('1vxznov16031kjxneqjk1rtofi6', clf, d))

print('Predictions of domain %s is [0 means legit and 1 dga]: ' **%**('lthmqglxwmrwex'), is\_dga('lthmqglxwmrwex', clf, d))

is\_dga('brandeis.edu', clf, d)

**Assess model accuracy with simple cross-validation**

* Calling the .predict() method on the clf with your training data X\_train and store the results in a variable called target\_pred.
* Using sklearn [metrics.accuracy\_score](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html) to determine your models accuracy. Detailed Instruction:
  + Using trained model to predict the labels of your test data X\_test. Run .predict() method on the clf with your test data X\_test and store the results in a variable called target\_pred..
  + Then calculate the accuracy using target\_test (which are the true labels/groundtruth) AND your models predictions on the test portion target\_pred as inputs. The advantage here is to see how your model performs on new data it has not been seen during the training phase. The fair approach here is a simple **cross-validation**!
* Print out the confusion matrix using [metrics.confusion\_matrix](http://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html)
* Use Yellowbrick to visualize the classification report and confusion matrix. (<http://www.scikit-yb.org/en/latest/examples/modelselect.html#common-metrics-for-evaluating-classifiers>)

*# fair approach: make prediction on test data portion*

target\_pred **=** clf**.**predict(feature\_matrix\_test)

print(metrics**.**accuracy\_score(target\_test, target\_pred))

print('Confusion Matrix\n', metrics**.**confusion\_matrix(target\_test, target\_pred))

*# Classification Report...neat summary*

print(metrics**.**classification\_report(target\_test, target\_pred, target\_names**=**['legit', 'dga']))

*# short-cut*

clf**.**score(feature\_matrix\_test, target\_test)

viz **=** ConfusionMatrix(clf)

viz**.**fit(feature\_matrix\_train, target\_train)

viz**.**score(feature\_matrix\_test, target\_test)

viz**.**show()

viz **=** ClassificationReport(clf)

viz**.**fit(feature\_matrix\_train, target\_train)

viz**.**score(feature\_matrix\_test, target\_test)

viz**.**poof()

**Assess model accuracy with k-fold cross-validation**

* Partition the dataset into *k* different subsets
* Create *k* different models by training on *k-1* subsets and testing on the remaining subsets
* Measure the performance on each of the models and take the average measure.

cvKFold **=** model\_selection**.**KFold(n\_splits**=**3, shuffle**=True**, random\_state**=**33)

cvKFold**.**get\_n\_splits(feature\_matrix)

scores **=** model\_selection**.**cross\_val\_score(clf, feature\_matrix, target, cv**=**cvKFold)

print(scores)

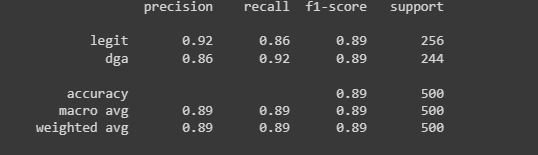
*# Get avergage score +- Standard Error (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.sem.html)*

**from** scipy.stats **import** sem

**def** mean\_score( scores ):

**return** "Mean score: {0:.3f} (+/- {1:.3f})"**.**format( np**.**mean(scores), sem( scores ))

print( mean\_score( scores))



**Visualizing Tree**

**from** IPython.core.display **import** Image

**import** pydotplus **as** pydot

dot\_data **=** io**.**StringIO()

tree**.**export\_graphviz(clf, out\_file**=**dot\_data,

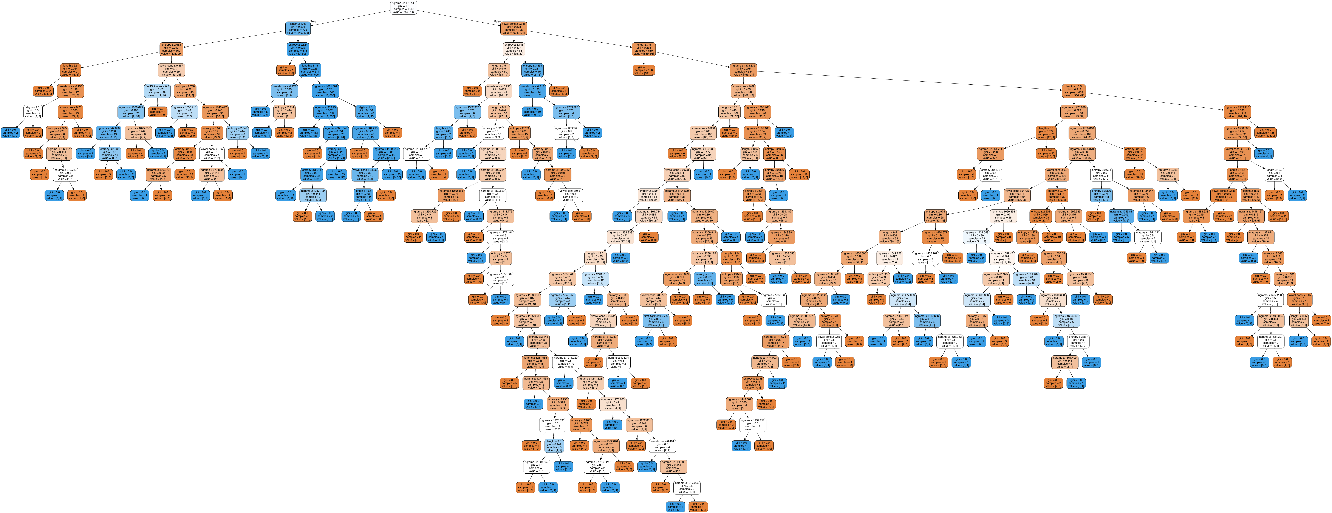
feature\_names**=**['length', 'digits', 'entropy', 'vowel-cons', 'firstDigitIndex','ngrams'],

filled**=True**, rounded**=True**,

special\_characters**=True**)

graph **=** pydot**.**graph\_from\_dot\_data(dot\_data**.**getvalue())

Image(graph**.**create\_png())



**Other Models**

* Support Vector Machine
* Random Forest
* K-Nearest Neighbors (<http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>)

**from** sklearn **import** svm

**from** sklearn.ensemble **import** RandomForestClassifier

**from** sklearn.neighbors **import** KNeighborsClassifier

*#Create the Random Forest Classifier*

random\_forest\_clf **=** RandomForestClassifier(n\_estimators**=**10,

max\_depth**=None**,

min\_samples\_split**=**2,

random\_state**=**0)

random\_forest\_clf **=** random\_forest\_clf**.**fit(feature\_matrix\_train, target\_train)

*#Next, create the SVM classifier*

svm\_classifier **=** svm**.**SVC()

svm\_classifier **=** svm\_classifier**.**fit(feature\_matrix\_train, target\_train)

*#Finally the knn*

knn\_clf **=** KNeighborsClassifier()

knn\_clf **=** knn\_clf**.**fit(feature\_matrix\_train, target\_train)

**Explain a Prediction**

**import** lime.lime\_tabular

explainer **=** lime**.**lime\_tabular**.**LimeTabularExplainer(feature\_matrix\_train,

feature\_names**=**['length', 'digits', 'entropy', 'vowel-cons', 'firstDigitIndex','ngrams'],

class\_names**=**['legit', 'isDGA'],

discretize\_continuous**=False**)

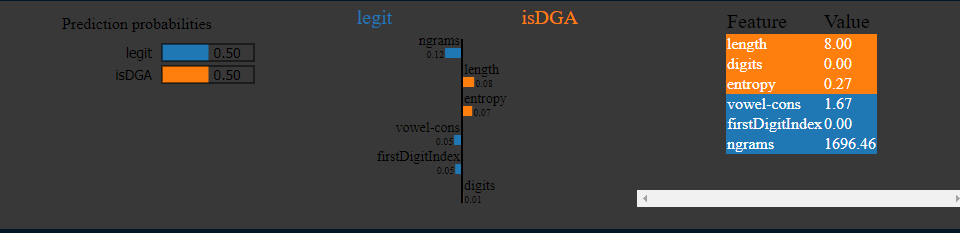
exp **=** explainer**.**explain\_instance(feature\_matrix\_test**.**iloc[9],

random\_forest\_clf**.**predict\_proba,

num\_features**=**6)

exp**.**show\_in\_notebook(show\_table**=True**, show\_all**=True**)

feature\_matrix\_test**.**iloc[5]



**Confusion Matrix**

!pip install scikit-plot

from sklearn.metrics import ConfusionMatrixDisplay

viz = ConfusionMatrixDisplay.from\_predictions(

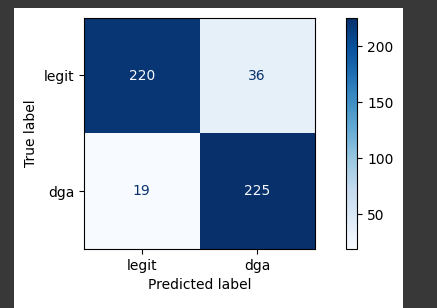
    target\_test, clf.predict(feature\_matrix\_test),

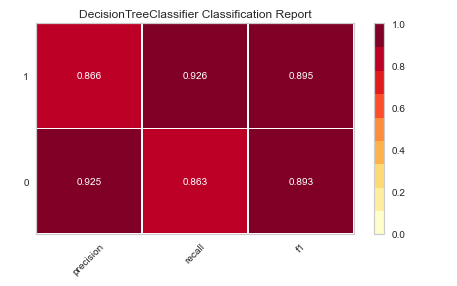
    display\_labels=['legit', 'dga'], cmap=plt.cm.Blues

)

viz.plot()

viz.show()





**Conclusion**

The confusion matrix and other related outputs like decision tree has been attached.