Fixed_Urls_Logit

October 14, 2020

1 Comparison of Logistic Regression on base URL vs full URL

1.1 Summary of experiment:

The Alexa dataset is taken as a proxy for "benign web sites." This is a faulty assumption as Alexa rankings just track popularity. Malicious web services (adware) can be found in Alexa, especially with browser extensions that inject ads into websites and sell user information. Additionally, in comparing Alexa-sourced URLs to other URLs, the Alexa URLs are just a host name (basedomain.com) instead of fully formed. This makes them statistically invalid as a comparison as they are strings with less complexity/entropy.

In order to screen out malicious URLs from Alexa, the host name for every listing was queried on Google – the search was "site:base_url base_url login". This allowed augmentation of the base_url for legitimate sites, allowing for phishing URLs and their respective page content (HTML with scripts and styles) to be compared to presumably benign URLs and page content. This approach fixed an issue with internal validity (some Alexa listings are not benign) and with construct validity (phishing URLs were previously being compared to host names).

In creating the HTML/document-level features, the Alexa set was augmented to include the HTML from the host name (the "home page") and a canonical URL (either a login page, or a popular page on that site). This allows for other research to be done in terms of A vs B testing.

This document shows the following:

1.1.1 Data sets used

Alexa top 1 million (from kaggle, outdated)

PhishTank, sample from Summer 2020

1.1.2 Hypotheses

1) Alexa as a data set performs with artificially high accuracy due to a construct error
2) Alexa has an internal validity issue in that some of its URLs are not benign
3) A mixture of login pages and arbitrary pages (whether home page, or deeper within a site) will help generalize phishing detection; treating benign pages as monolithic leads to external validity issues.
4) Word vectors are inadequate as there are no natural word breaks in URLs the way we have spaces/periods in modern languages. Statistical validity issue.

5) Shannon Entropy (string complexity of URL) is inadequate for forecasting

1.1.3 Logistic Regression using word vector of URL:

Base URLs (Alexa) compared to Full URLs (Phishtank) - 93%

Base URLs compared - 89% accuracy

Full URLs compared - **84% accuracy**

1.1.4 SoftMax (Multivate Logit) on 3 features – number of periods, presence of @ or - in URL, and URL length

Combined URLs - **86%**

Base URLs - 84%

Full URLs - **71%**

The main take-away from this is that accuracy can be artificially inflated if there is a problem with construct validity; more features remain to be seen. Additionally, these are URL-level, not page content or domain-level, so there is missing information.

The longer the string, the more complexity; but there are salient syntactic features that these three indicators have not looked at. Additionally, semantic markers (popular brand names, topics that appear often in phishing attempts) are not preserved by using a heuristic approach.

1.1.5 SoftMax on Base+Full 9 features

Combined **86.3%**

Base 83.88%

Full 71.8%

1.1.6 SoftMax on Base+Full – Just Entropy

Base URL 74%

Full URL 67.07%

Combined 75.51%

1.1.7 Decision Tree with Base+Full URL features

3 features (periods, special chars, length) - **85.1%**

Expanded features: Base+Full - **85.3%**

Base - **85.3%**

Full - **74.59%**

1.1.8 SVM

Unable to stabilize results at this time

1.1.9 MLP

Base 0.84693

Full 0.79538

Base URL Shannon Entropy 0.76315

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

#import functools
#import numpy as np
#import tensorflow as tf
```

[2]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split

from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
```

[3]:
```
# globally specify solver to suppress warning
LogisticRegression(solver='lbfgs')
```

[3]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

2 Preprocessing section (csv-level)

[4]:
```
# data warehousing has been moved to a separate pipeline
```

3 Load data / do logistic reg

[5]:
```
filepath_dict = {#'ptank_orig':   'quick2000.csv', # original dataset with just␣
 ↪url+flag variables
                 #'ptankalexa':   'fixed2000.csv'
                'ptankalexa':   'nine2000.csv'
                }

df_list = []
for source, filepath in filepath_dict.items():
    df = pd.read_csv(filepath, sep=',')
    #df = pd.read_csv(filepath, names=['url', 'label'], sep=',')
    #data = pd.read_csv(ptank3)
    df['source'] = source  # Add another column filled with the source name
    df_list.append(df)

df = pd.concat(df_list)
print(df.iloc[0])
dfcolnames = df.columns
```

```
url                  https://wdestaques13.a-semana-especial-chegou…
flag                                                                 1
full_url             https://wdestaques13.a-semana-especial-chegou…
base_url             https://wdestaques13.a-semana-especial-chegou.com
base_num_periods                                                     2
full_num_periods                                                     2
base_spec_symbols                                                 True
full_spec_symbols                                                 True
base_length                                                         49
full_length                                                        135
ip                                                                   0
full_anchors                                                         0
base_anchors                                                         0
full_params                                                          0
base_params                                                          0
full_queries                                                         0
base_queries                                                         0
full_digits                                                         18
base_digits                                                          2
full_entropy                                                  -5.03622
base_entropy                                                  -4.24749
source                                                      ptankalexa
Name: 0, dtype: object
```

[6]:
```
print(dfcolnames)
```

```
Index(['url', 'flag', 'full_url', 'base_url', 'base_num_periods',
       'full_num_periods', 'base_spec_symbols', 'full_spec_symbols',
       'base_length', 'full_length', 'ip', 'full_anchors', 'base_anchors',
       'full_params', 'base_params', 'full_queries', 'base_queries',
       'full_digits', 'base_digits', 'full_entropy', 'base_entropy', 'source'],
      dtype='object')
```

[7]:
```
df_ptank = df[df['source'] == 'ptankalexa']
```

[8]:
```
url_string = df_ptank['base_url'].values # swap 'url' with 'base_url'
```

[9]:
```
y = df_ptank['flag'].valuesy = df_ptank['flag'].values
```

4 Just base url (anything after http[s]:// up to the TLD

[10]:

```
url_train, url_test, y_train, y_test = train_test_split(url_string,
                                                                    y,
                                                                    test_size=0.
 ↪25,
                                                                   ␣
 ↪random_state=1000)
```

[11]:

```
# from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(url_train)

X_train = vectorizer.transform(url_train)
X_test  = vectorizer.transform(url_test)
X_train
```

[11]:

```
<1396x1500 sparse matrix of type '<class 'numpy.int64'>'
        with 5037 stored elements in Compressed Sparse Row format>
```

[12]:

```
# specify a solver to suppress warning
LogisticRegression(solver='lbfgs')
```

[12]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[13]:

```
classifier = LogisticRegression()
classifier.fit(X_train, y_train)
score = classifier.score(X_test, y_test)


print("Accuracy:", score)
```

```
Accuracy: 0.8927038626609443
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

Changing ‘url’ to ‘base_url’ decreases accuracy, confirmation that information reduction decreases logistic regression accuracy

5 Full URL Comparison

[14]:

```
url_string = df_ptank['full_url'].values # swap 'url' with 'base_url'
y = df_ptank['flag'].values
url_train, url_test, y_train, y_test = train_test_split(url_string,
                                                                    y,
                                                                    test_size=0.
 ↪25,
                                                                   ␣
 ↪random_state=1000)
```

[15]:

```
# from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(url_train)

X_train = vectorizer.transform(url_train)
X_test  = vectorizer.transform(url_test)
X_train
```

[15]:

```
<1396x3339 sparse matrix of type '<class 'numpy.int64'>'
        with 10162 stored elements in Compressed Sparse Row format>
```

[16]:

```
# specify a solver to suppress warning
LogisticRegression(solver='lbfgs')
```

[16]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[17]:

```
from sklearn.linear_model import LogisticRegression


classifier = LogisticRegression()
classifier.fit(X_train, y_train)
score = classifier.score(X_test, y_test)


print("Accuracy:", score)
```

```
Accuracy: 0.8454935622317596
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

6 Softmax (Multivariate Logit)

[18]:

```
# ignore/delete after scavenging:


url_string = df_ptank['url'].values # swap 'url' with 'base_url'


y = df_ptank['flag'].values


url_train, url_test, y_train, y_test = train_test_split(url_string,
                                                                    y,
                                                                    test_size=0.
 ↪25,
                                                                   ␣
 ↪random_state=1000)

# from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
vectorizer.fit(url_train)

X_train = vectorizer.transform(url_train)
X_test  = vectorizer.transform(url_test)
#X_train

# specify a solver to suppress warning
LogisticRegression(solver='lbfgs')

from sklearn.linear_model import LogisticRegression


classifier = LogisticRegression()
classifier.fit(X_train, y_train)
score = classifier.score(X_test, y_test)


print("Accuracy:", score)
```

```
Accuracy: 0.9334763948497854
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

[19]:

```
url_string = df_ptank['url'].values

vectorizer = CountVectorizer()
vectorizer.fit(url_string)


#url_vec = vectorizer.transform(url_string)
url_vec = vectorizer.fit_transform(url_string).toarray()
```

[20]:

```
print(url_vec)
```

```
[[0 0 0 … 0 0 0]
 [0 0 0 … 0 0 0]
 [0 0 0 … 0 0 0]
 …
 [0 0 0 … 0 0 0]
 [0 0 0 … 0 0 0]
 [0 0 0 … 0 0 0]]
```

[ ]:

```

```

[21]:

```
X = df[['base_num_periods', 'base_spec_symbols','base_length']] # convert␣
 ↪'base_url',  to vector as in previous example
y = df['flag']
lr = LogisticRegression()
lr.fit(X, y)
preds = lr.predict(X)
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

[22]:

```
print(lr.score(X, y))
```

```
0.841031149301826
```

[23]:

```
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score

print(confusion_matrix(y,preds))
print(classification_report(y,preds))
print(accuracy_score(y, preds))
```

```
[[747 115]
 [181 819]]
              precision    recall  f1-score   support
```

```
           0       0.80      0.87      0.83       862
           1       0.88      0.82      0.85      1000

    accuracy                           0.84      1862
   macro avg       0.84      0.84      0.84      1862
weighted avg       0.84      0.84      0.84      1862

0.841031149301826
```

[24]:

```
X = df[['full_num_periods', 'full_spec_symbols','full_length']]
y = df['flag']
lr = LogisticRegression()
lr.fit(X, y)
preds = lr.predict(X)
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

[25]:

```
print(lr.score(X, y))
```

```
0.7062298603651987
```

[ ]:

```

```

[26]:

```
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score

print(confusion_matrix(y,preds))
print(classification_report(y,preds))
print(accuracy_score(y, preds))
```

```
[[697 165]
 [382 618]]
              precision    recall  f1-score   support

           0       0.65      0.81      0.72       862
           1       0.79      0.62      0.69      1000

    accuracy                           0.71      1862
   macro avg       0.72      0.71      0.71      1862
weighted avg       0.72      0.71      0.70      1862

0.7062298603651987
```

7 Full+Base Features Combined

[27]:

```
X = df[['base_num_periods',␣
 ↪'base_spec_symbols','base_length','full_num_periods',␣
 ↪'full_spec_symbols','full_length']]
y = df['flag']
lr = LogisticRegression()
lr.fit(X, y)
preds = lr.predict(X)
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

[28]:

```
print(lr.score(X, y))
```

```
0.8668098818474759
```

[29]:

```
#from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score

print(confusion_matrix(y,preds))
print(classification_report(y,preds))
print(accuracy_score(y, preds))
```

```
[[772  90]
 [158 842]]
              precision    recall  f1-score   support

           0       0.83      0.90      0.86       862
           1       0.90      0.84      0.87      1000

    accuracy                           0.87      1862
   macro avg       0.87      0.87      0.87      1862
weighted avg       0.87      0.87      0.87      1862

0.8668098818474759
```

8 Softmax with 9 Features (Full+Base)

[30]:

```
drop_cols = ['url', 'flag', 'full_url', 'base_url','source', 'ip']

df2 = df.drop(drop_cols,axis=1)
```

[31]:

```
base_features = ['base_num_periods', 'base_spec_symbols', 'base_length',␣
 ↪'base_anchors', 'base_params', 'base_queries', 'base_digits', 'base_entropy']
full_features = ['full_num_periods', 'full_spec_symbols', 'full_length',␣
 ↪'full_anchors', 'full_params', 'full_queries', 'full_digits', 'full_entropy']


df2_full = df2.drop(base_features,axis=1)
df2_base = df2.drop(full_features,axis=1)

X_full = df2_full
y = df['flag']

X_base = df2_base
y = df['flag']
```

[32]:

```
# new df that drops IP, url, base_url, full_url


#urls_features = ['']

print(df2.columns)

X = df2 #df[['base_num_periods', 'base_spec_symbols','base_length']] # convert␣
 ↪'base_url',  to vector as in previous example
y = df['flag']
lr = LogisticRegression()
lr.fit(X, y)
preds = lr.predict(X)


print(lr.score(X, y))


from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score

print(confusion_matrix(y,preds))
print(classification_report(y,preds))
print(accuracy_score(y, preds))
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

```
Index(['base_num_periods', 'full_num_periods', 'base_spec_symbols',
       'full_spec_symbols', 'base_length', 'full_length', 'full_anchors',
       'base_anchors', 'full_params', 'base_params', 'full_queries',
       'base_queries', 'full_digits', 'base_digits', 'full_entropy',
       'base_entropy'],
      dtype='object')
0.8635875402792696
[[775  87]
 [167 833]]
              precision    recall  f1-score   support

           0       0.82      0.90      0.86       862
           1       0.91      0.83      0.87      1000

    accuracy                           0.86      1862
   macro avg       0.86      0.87      0.86      1862
weighted avg       0.87      0.86      0.86      1862

0.8635875402792696
```

[33]:

```
# new df that drops IP, url, base_url, full_url


#urls_features = ['']

print(df2_base.columns)

X = df2_base #df[['base_num_periods', 'base_spec_symbols','base_length']] #␣
 ↪convert 'base_url',  to vector as in previous example
y = df['flag']
lr = LogisticRegression()
lr.fit(X_base, y)
preds = lr.predict(X_base)


print(lr.score(X_base, y))


from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score

print(confusion_matrix(y,preds))
print(classification_report(y,preds))
print(accuracy_score(y, preds))
```

```
Index(['base_num_periods', 'base_spec_symbols', 'base_length', 'base_anchors',
       'base_params', 'base_queries', 'base_digits', 'base_entropy'],
      dtype='object')
0.8388829215896885
[[761 101]
 [199 801]]
```

```
              precision    recall  f1-score   support

           0       0.79      0.88      0.84       862
           1       0.89      0.80      0.84      1000

    accuracy                           0.84      1862
   macro avg       0.84      0.84      0.84      1862
weighted avg       0.84      0.84      0.84      1862

0.8388829215896885
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

[34]:

```
# new df that drops IP, url, base_url, full_url



#urls_features = ['']

print(df2_full.columns)

X = df2_full #df[['base_num_periods', 'base_spec_symbols','base_length']] #␣
 ↪convert 'base_url',  to vector as in previous example
y = df['flag']
lr = LogisticRegression()
lr.fit(X, y)
preds = lr.predict(X)


print(lr.score(X_full, y))


from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score

print(confusion_matrix(y,preds))
print(classification_report(y,preds))
print(accuracy_score(y, preds))
```

```
Index(['full_num_periods', 'full_spec_symbols', 'full_length', 'full_anchors',
       'full_params', 'full_queries', 'full_digits', 'full_entropy'],
      dtype='object')
0.7180451127819549
[[685 177]
 [348 652]]
              precision    recall  f1-score   support
```

```
           0       0.66      0.79      0.72       862
           1       0.79      0.65      0.71      1000

    accuracy                           0.72      1862
   macro avg       0.72      0.72      0.72      1862
weighted avg       0.73      0.72      0.72      1862

0.7180451127819549
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

9 SoftMax - Just Shannon Entropy of URL

9.1 Base URL entropy

[35]:
```
# new df that drops IP, url, base_url, full_url


#urls_features = ['']




X = df[['base_entropy']] #df[['base_num_periods',␣
 ↪'base_spec_symbols','base_length']] # convert 'base_url',  to vector as in␣
 ↪previous example
y = df['flag']
lr = LogisticRegression()
lr.fit(X, y)
preds = lr.predict(X)


print(lr.score(X, y))


from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score

print(confusion_matrix(y,preds))
print(classification_report(y,preds))
print(accuracy_score(y, preds))
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
```

```
  FutureWarning)

0.7400644468313641
[[588 274]
 [210 790]]
              precision    recall  f1-score   support

           0       0.74      0.68      0.71       862
           1       0.74      0.79      0.77      1000

    accuracy                           0.74      1862
   macro avg       0.74      0.74      0.74      1862
weighted avg       0.74      0.74      0.74      1862

0.7400644468313641
```

9.2 Full URL entropy

[36]:

```
# new df that drops IP, url, base_url, full_url


#urls_features = ['']




X = df[['full_entropy']] #df[['base_num_periods',␣
 ↪'base_spec_symbols','base_length']] # convert 'base_url',  to vector as in␣
 ↪previous example
y = df['flag']
lr = LogisticRegression()
lr.fit(X, y)
preds = lr.predict(X)


print(lr.score(X, y))


from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score

print(confusion_matrix(y,preds))
print(classification_report(y,preds))
print(accuracy_score(y, preds))
```

```
0.6707841031149302
[[542 320]
 [293 707]]
              precision    recall  f1-score   support
```

```
           0       0.65      0.63      0.64       862
           1       0.69      0.71      0.70      1000

    accuracy                           0.67      1862
   macro avg       0.67      0.67      0.67      1862
weighted avg       0.67      0.67      0.67      1862

0.6707841031149302
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

9.3 Combined Full+Base URL entropy

[37]:

```
# new df that drops IP, url, base_url, full_url


#urls_features = ['']




X = df[['base_entropy','full_entropy']] #df[['base_num_periods',␣
 ↪'base_spec_symbols','base_length']] # convert 'base_url',  to vector as in␣
 ↪previous example
y = df['flag']
lr = LogisticRegression()
lr.fit(X, y)
preds = lr.predict(X)


print(lr.score(X, y))


from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score

print(confusion_matrix(y,preds))
print(classification_report(y,preds))
print(accuracy_score(y, preds))
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a
solver to silence this warning.
  FutureWarning)
```

```
0.7551020408163265
[[601 261]
 [195 805]]
              precision    recall  f1-score   support

           0       0.76      0.70      0.72       862
           1       0.76      0.81      0.78      1000

    accuracy                           0.76      1862
   macro avg       0.76      0.75      0.75      1862
weighted avg       0.76      0.76      0.75      1862

0.7551020408163265
```

[ ]:

```

```

10 D-Trees

[ ]:

```

```

[38]:

```
#predictor_columns = data2.columns
#d = data2[predictor_columns]
#x, y = d[predictor_columns], data[vt_class]

# remove flag from what goes into x
```

[39]:

```
# make new df with urls dropped
drop_cols = ['url', 'flag', 'full_url', 'base_url','source', 'ip']

#urls_features = ['']

df2 = df.drop(drop_cols,axis=1)
```

11 All Features (Derived from Base + Full URL)

[40]:

```
from sklearn import tree
from sklearn.metrics import accuracy_score

from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz

from IPython.display import Image

import pydotplus
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\externals\six.py:31:
DeprecationWarning: The module is deprecated in version 0.21 and will be removed
in version 0.23 since we've dropped support for Python 2.7. Please rely on the
official version of six (https://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

[41]:

```
print(df.columns)
#df2 = df[df.columns]
x, y = df2, df['flag']
```

```
Index(['url', 'flag', 'full_url', 'base_url', 'base_num_periods',
       'full_num_periods', 'base_spec_symbols', 'full_spec_symbols',
       'base_length', 'full_length', 'ip', 'full_anchors', 'base_anchors',
       'full_params', 'base_params', 'full_queries', 'base_queries',
       'full_digits', 'base_digits', 'full_entropy', 'base_entropy', 'source'],
      dtype='object')
```

[42]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 100)
```

[ ]:

```

```

[43]:

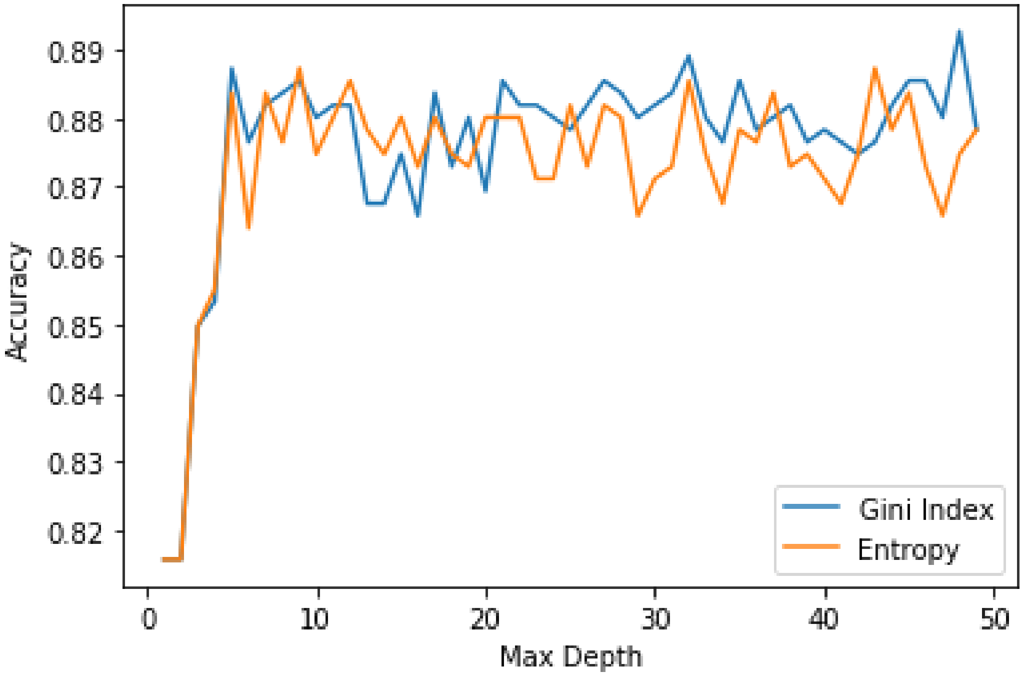
```
maxd, gini, entropy = [], [], []

iterations = 50
```

[44]:

```
#maxd, gini, entropy = [], [], []
for i in range(1,iterations):
    ###
    dtree = tree.DecisionTreeClassifier(criterion='gini', max_depth=i)
    dtree.fit(X_train, y_train)
    pred = dtree.predict(X_test)
    gini.append(accuracy_score(y_test, pred))

    ####
    dtree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=i)
    dtree.fit(X_train, y_train)
    pred = dtree.predict(X_test)
    entropy.append(accuracy_score(y_test, pred))

    ####
    maxd.append(i)

####
```

```
d = pd.DataFrame({'gini':pd.Series(gini), 'entropy':pd.Series(entropy),␣
 ↪'max_depth':pd.Series(maxd)})
# visualizing changes in parameters
plt.plot('max_depth','gini', data=d, label='Gini Index')
plt.plot('max_depth','entropy', data=d, label='Entropy')
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.legend()
```

```
[44]: <matplotlib.legend.Legend at 0x18153c02f08>
```



```
[45]: DT = tree.DecisionTreeClassifier(criterion="gini", max_depth=4)

      ##fit decision tree model with training data
      DT.fit(X_train, y_train)

      ##test data prediction
      DT_expost_preds = DT.predict(X_test)
```

```
[46]: print(confusion_matrix(y_test, DT_expost_preds))
      print(classification_report(y_test,DT_expost_preds))
      print(accuracy_score(y_test, DT_expost_preds))
```

```
[[220  21]
 [ 61 257]]
```

```
              precision    recall  f1-score   support

           0       0.78      0.91      0.84       241
           1       0.92      0.81      0.86       318

    accuracy                           0.85       559
   macro avg       0.85      0.86      0.85       559
weighted avg       0.86      0.85      0.85       559

0.8533094812164580
```

[47]:

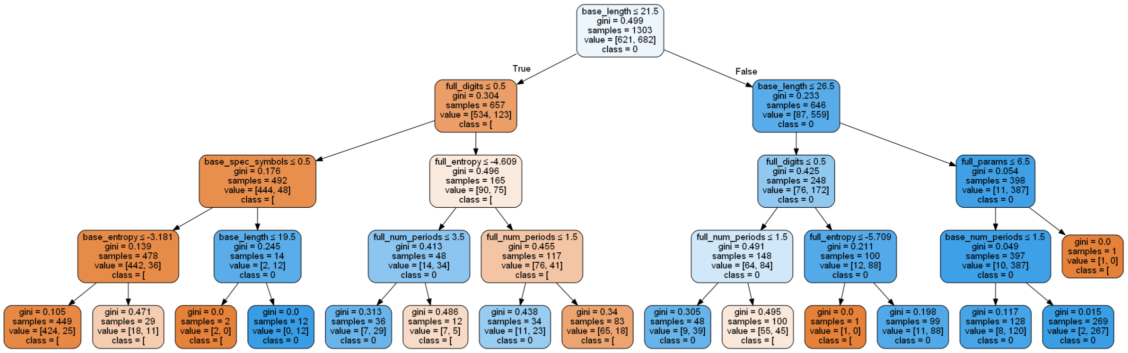
```
print(DT_expost_preds)
```

```
[0 1 1 0 0 1 1 1 0 1 1 0 1 1 0 0 0 1 1 1 0 1 0 1 1 0 1 0 0 0 0 1 1 1 1 1 0
 1 1 0 0 1 0 0 0 1 0 0 1 1 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0 1 0 1 1 1 1 0 1 1
 0 0 1 1 0 0 1 1 0 0 0 0 1 1 1 0 0 0 1 0 0 0 0 1 1 0 0 0 1 0 0 1 0 0 1 1 0
 1 0 1 1 1 1 0 1 0 1 0 1 0 1 0 0 1 0 1 0 0 1 1 1 0 0 0 1 1 1 0 1 1 0 0 1 1
 1 0 0 1 1 1 0 1 1 0 1 0 0 0 0 0 0 1 1 0 1 1 0 0 0 1 1 0 1 1 0 0 0 0 1 1 1
 0 1 0 0 0 1 0 1 0 1 0 0 0 1 1 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1 0 1 0 1 1 0
 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 1 1 0 1 0 1 0 0 0 1 1 1 1 1 0 1 0 0 0 0 1 0
 1 1 0 0 1 1 1 0 1 1 0 0 1 0 0 1 1 0 0 0 0 0 0 1 0 0 0 1 0 1 0 1 0 1 1 0 1
 1 1 0 0 0 0 1 0 1 0 1 1 1 0 0 0 1 1 0 1 1 1 1 1 0 0 0 0 0 1 1 0 0 0 0 0 0
 1 0 0 1 1 0 0 1 0 1 1 1 1 0 0 1 0 0 1 1 1 0 0 1 1 0 0 1 0 0 0 1 0 0 0 1 1
 0 1 1 0 1 1 0 0 1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1
 0 1 0 1 0 0 1 1 1 0 0 0 1 0 1 0 1 1 1 0 1 0 0 1 0 1 0 0 0 1 1 0 1 1 0 1 1
 1 0 0 1 1 0 0 0 0 1 0 1 1 0 0 1 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 0 1 0 0
 0 0 0 1 1 0 0 0 1 0 1 1 1 1 0 0 1 1 0 1 1 1 0 1 1 0 0 1 1 0 0 0 0 0 0 1 1
 1 1 1 1 0 1 0 0 0 0 1 1 1 1 1 0 0 0 0 1 1 1 1 1 1 0 1 0 1 0 0 0 0 1 0 1 0
 0 0 1 0]
```

[48]:

```
dot_data = StringIO()
export_graphviz(DT, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names=X_train.columns,␣
 ↪class_names=str(DT.classes_))
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())


Image(graph.create_png())
```

[48]:



12 Just Base URL features

[49]:
```
x, y = df2_base, df['flag']
```

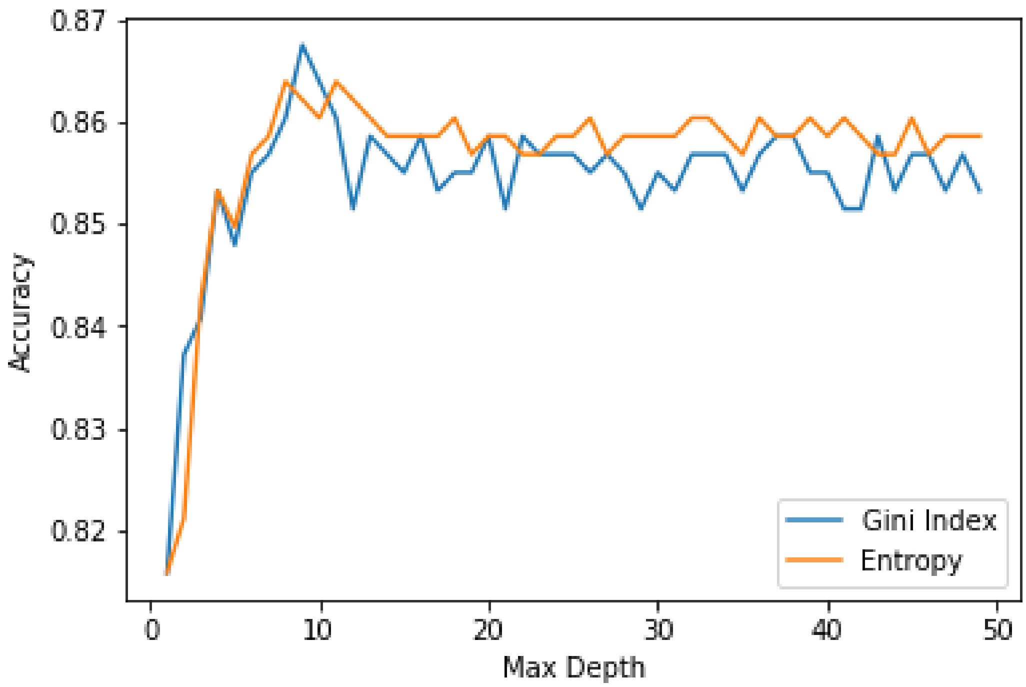
[50]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 100)
```

[51]:
```
maxd, gini, entropy = [], [], []

iterations = 50
```

[52]:
```
#maxd, gini, entropy = [], [], []
for i in range(1,iterations):
    ###
    dtree = tree.DecisionTreeClassifier(criterion='gini', max_depth=i)
    dtree.fit(X_train, y_train)
    pred = dtree.predict(X_test)
    gini.append(accuracy_score(y_test, pred))

    ####
    dtree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=i)
    dtree.fit(X_train, y_train)
    pred = dtree.predict(X_test)
    entropy.append(accuracy_score(y_test, pred))

    ####
    maxd.append(i)

####
```

```
d = pd.DataFrame({'gini':pd.Series(gini), 'entropy':pd.Series(entropy),␣
 ↪'max_depth':pd.Series(maxd)})
# visualizing changes in parameters
plt.plot('max_depth','gini', data=d, label='Gini Index')
plt.plot('max_depth','entropy', data=d, label='Entropy')
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.legend()
```

```
[52]: <matplotlib.legend.Legend at 0x181540973c8>
```



```
[53]: DT = tree.DecisionTreeClassifier(criterion="gini", max_depth=4)

      ##fit decision tree model with training data
      DT.fit(X_train, y_train)

      ##test data prediction
      DT_expost_preds = DT.predict(X_test)
```

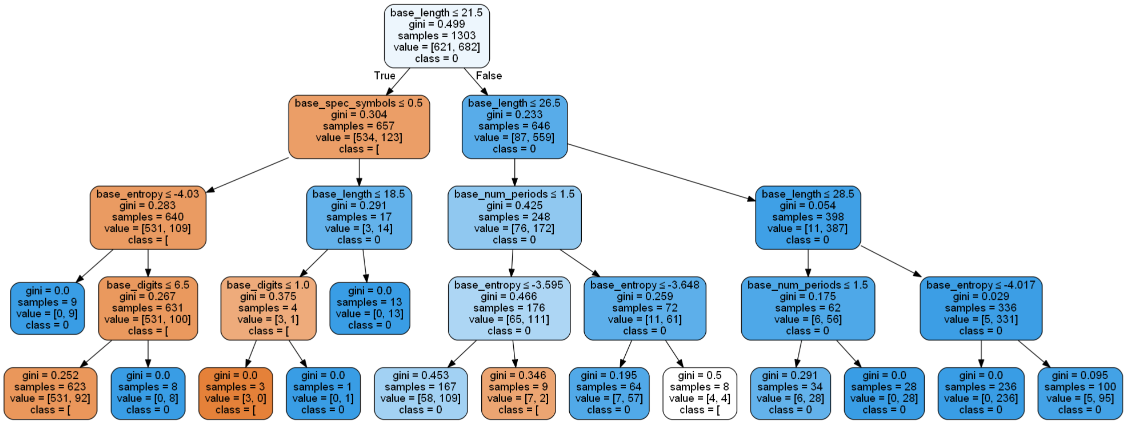
```
[54]: print(confusion_matrix(y_test, DT_expost_preds))
      print(classification_report(y_test,DT_expost_preds))
      print(accuracy_score(y_test, DT_expost_preds))
```

```
[[208  33]
 [ 50 268]]
```

```
              precision    recall  f1-score   support

           0       0.81      0.86      0.83       241
           1       0.89      0.84      0.87       318

    accuracy                           0.85       559
   macro avg       0.85      0.85      0.85       559
weighted avg       0.85      0.85      0.85       559

0.851520572450805
```

[55]:
```
dot_data = StringIO()
export_graphviz(DT, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names=X_train.columns,␣
 ↪class_names=str(DT.classes_))
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())


Image(graph.create_png())
```

[55]:


base_length ≤ 21.5 gini = 0.499 samples = 1303 value = [621, 682] class = 0
True
False
base_spec_symbols ≤ 0.5 gini = 0.304 samples = 657 value = [534, 123] class = [
base_length ≤ 26.5 gini = 0.233 samples = 646 value = [87, 559] class = 0
base_entropy ≤ -4.03 gini = 0.283 samples = 640 value = [531, 109] class = [
base_length ≤ 18.5 gini = 0.291 samples = 17 value = [3, 14] class = 0
base_num_periods ≤ 1.5 gini = 0.425 samples = 248 value = [76, 172] class = 0
base_length ≤ 28.5 gini = 0.054 samples = 398 value = [11, 387] class = 0
gini = 0.0 samples = 9 value = [0, 9] class = 0
base_digits ≤ 6.5 gini = 0.267 samples = 631 value = [531, 100] class = [
base_digits ≤ 1.0 gini = 0.375 samples = 4 value = [3, 1] class = [
gini = 0.0 samples = 13 value = [0, 13] class = 0
base_entropy ≤ -3.595 gini = 0.466 samples = 176 value = [65, 111] class = 0
base_entropy ≤ -3.648 gini = 0.259 samples = 72 value = [11, 61] class = 0
base_num_periods ≤ 1.5 gini = 0.175 samples = 62 value = [6, 56] class = 0
base_entropy ≤ -4.017 gini = 0.029 samples = 336 value = [5, 331] class = 0
gini = 0.252 samples = 623 value = [531, 92] class = [
gini = 0.0 samples = 8 value = [0, 8] class = 0
gini = 0.0 samples = 3 value = [3, 0] class = [
gini = 0.0 samples = 1 value = [0, 1] class = 0
gini = 0.453 samples = 167 value = [58, 109] class = 0
gini = 0.346 samples = 9 value = [7, 2] class = [
gini = 0.195 samples = 64 value = [7, 57] class = 0
gini = 0.5 samples = 8 value = [4, 4] class = [
gini = 0.291 samples = 34 value = [6, 28] class = 0
gini = 0.0 samples = 28 value = [0, 28] class = 0
gini = 0.0 samples = 236 value = [0, 236] class = 0
gini = 0.095 samples = 100 value = [5, 95] class = 0

13 Just Full URL features

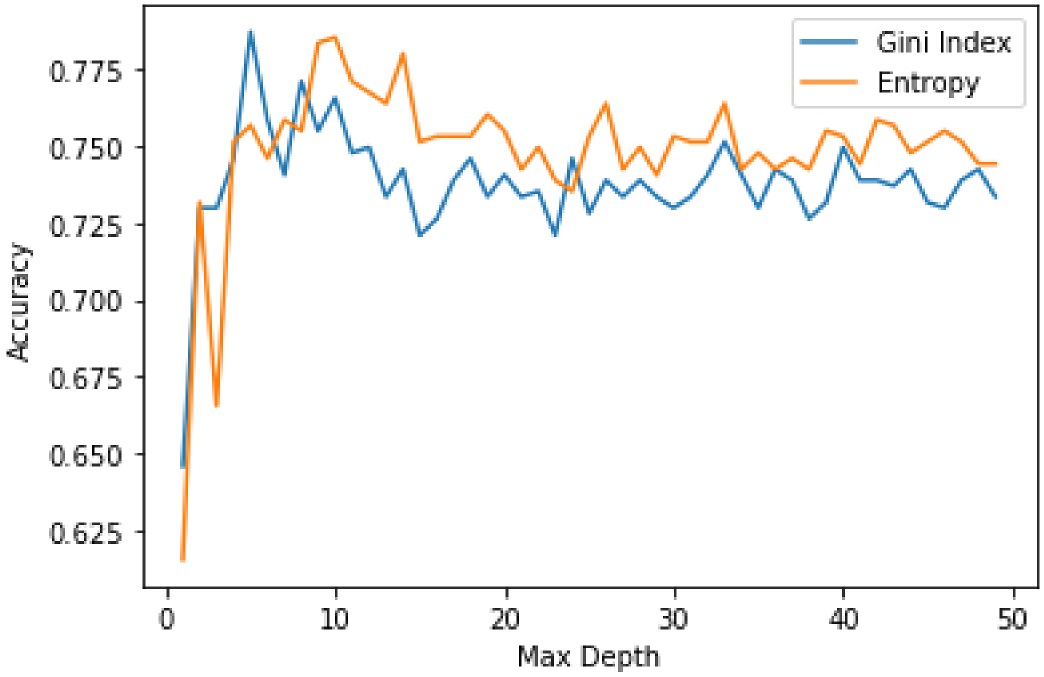
[56]:
```
x, y = df2_full, df['flag']
```

[57]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 100)
```

[58]:
```
maxd, gini, entropy = [], [], []

iterations = 50
```

[59]:
```
#maxd, gini, entropy = [], [], []
for i in range(1,iterations):
    ###
    dtree = tree.DecisionTreeClassifier(criterion='gini', max_depth=i)
    dtree.fit(X_train, y_train)
    pred = dtree.predict(X_test)
    gini.append(accuracy_score(y_test, pred))

    ####
    dtree = tree.DecisionTreeClassifier(criterion='entropy', max_depth=i)
    dtree.fit(X_train, y_train)
    pred = dtree.predict(X_test)
    entropy.append(accuracy_score(y_test, pred))

    ####
    maxd.append(i)

####
d = pd.DataFrame({'gini':pd.Series(gini), 'entropy':pd.Series(entropy),␣
 ↪'max_depth':pd.Series(maxd)})
# visualizing changes in parameters
plt.plot('max_depth','gini', data=d, label='Gini Index')
plt.plot('max_depth','entropy', data=d, label='Entropy')
plt.xlabel('Max Depth')
plt.ylabel('Accuracy')
plt.legend()
```

[59]:
```
<matplotlib.legend.Legend at 0x1815076f908>
```



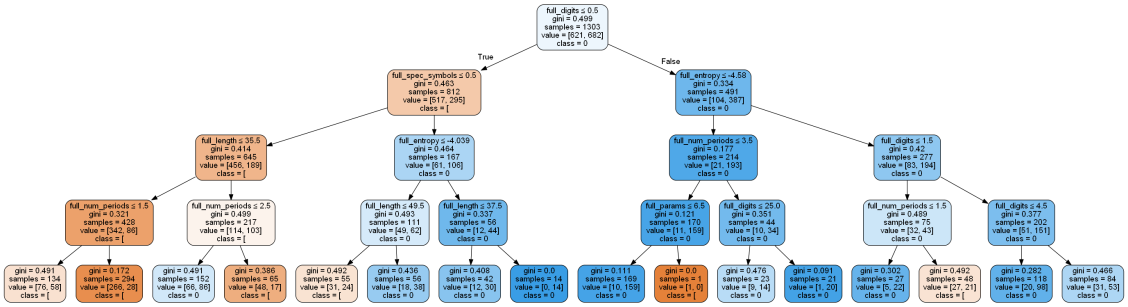
```
[60]: DT = tree.DecisionTreeClassifier(criterion="gini", max_depth=4)

      ##fit decision tree model with training data
      DT.fit(X_train, y_train)

      ##test data prediction
      DT_expost_preds = DT.predict(X_test)
```

```
[61]: print(confusion_matrix(y_test, DT_expost_preds))
      print(classification_report(y_test,DT_expost_preds))
      print(accuracy_score(y_test, DT_expost_preds))
```

```
[[173  68]
 [ 74 244]]
              precision    recall  f1-score   support

           0       0.70      0.72      0.71       241
           1       0.78      0.77      0.77       318

    accuracy                           0.75       559
   macro avg       0.74      0.74      0.74       559
weighted avg       0.75      0.75      0.75       559

0.7459749552772809
```

[62]:
```
dot_data = StringIO()
export_graphviz(DT, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names=X_train.columns,␣
 ↪class_names=str(DT.classes_))
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())


Image(graph.create_png())
```

[62]:


full_digits ≤ 0.5 gini = 0.499 samples = 1303 value = [621, 682] class = 0
True
False
full_spec_symbols ≤ 0.5 gini = 0.463 samples = 812 value = [517, 295] class = [
full_entropy ≤ -4.58 gini = 0.334 samples = 491 value = [104, 387] class = 0
full_length ≤ 35.5 gini = 0.414 samples = 645 value = [456, 189] class = [
full_entropy ≤ -4.039 gini = 0.464 samples = 167 value = [61, 106] class = 0
full_num_periods ≤ 3.5 gini = 0.177 samples = 214 value = [21, 193] class = 0
full_digits ≤ 1.5 gini = 0.42 samples = 277 value = [83, 194] class = 0
full_num_periods ≤ 1.5 gini = 0.321 samples = 428 value = [342, 86] class = [
full_num_periods ≤ 2.5 gini = 0.499 samples = 217 value = [114, 103] class = [
full_length ≤ 49.5 gini = 0.493 samples = 111 value = [49, 62] class = 0
full_length ≤ 37.5 gini = 0.337 samples = 56 value = [12, 44] class = 0
full_params ≤ 6.5 gini = 0.121 samples = 170 value = [11, 159] class = 0
full_digits ≤ 25.0 gini = 0.351 samples = 44 value = [10, 34] class = 0
full_num_periods ≤ 1.5 gini = 0.489 samples = 75 value = [32, 43] class = 0
full_digits ≤ 4.5 gini = 0.377 samples = 202 value = [51, 151] class = 0
gini = 0.491 samples = 134 value = [76, 58] class = [
gini = 0.172 samples = 294 value = [266, 28] class = [
gini = 0.491 samples = 152 value = [66, 86] class = 0
gini = 0.386 samples = 65 value = [48, 17] class = [
gini = 0.492 samples = 55 value = [31, 24] class = [
gini = 0.436 samples = 56 value = [18, 38] class = 0
gini = 0.408 samples = 42 value = [12, 30] class = 0
gini = 0.0 samples = 14 value = [0, 14] class = 0
gini = 0.111 samples = 169 value = [10, 159] class = 0
gini = 0.0 samples = 1 value = [1, 0] class = [
gini = 0.476 samples = 23 value = [9, 14] class = 0
gini = 0.091 samples = 21 value = [1, 20] class = 0
gini = 0.302 samples = 27 value = [5, 22] class = 0
gini = 0.492 samples = 48 value = [27, 21] class = [
gini = 0.282 samples = 118 value = [20, 98] class = 0
gini = 0.466 samples = 84 value = [31, 53] class = 0

[ ]:
```

```

14 Support Vector Machine (SVM)

https://scikit-learn.org/stable/modules/svm.html

[ ]:
```

```

[ ]:
```

```

[63]:
```
# make new df with urls dropped
drop_cols = ['url', 'flag', 'full_url', 'base_url','source', 'ip']

#urls_features = ['']

df2 = df.drop(drop_cols,axis=1)
print(df.columns)
#df2 = df[df.columns]
X, y = df2, df['flag']
```

```
Index(['url', 'flag', 'full_url', 'base_url', 'base_num_periods',
       'full_num_periods', 'base_spec_symbols', 'full_spec_symbols',
       'base_length', 'full_length', 'ip', 'full_anchors', 'base_anchors',
       'full_params', 'base_params', 'full_queries', 'base_queries',
       'full_digits', 'base_digits', 'full_entropy', 'base_entropy', 'source'],
      dtype='object')
```

[64]:
```
#drop_cols = ['url', 'flag', 'full_url', 'base_url','source', 'ip']

#df2 = df.drop(drop_cols,axis=1)



base_features = ['base_num_periods', 'base_spec_symbols', 'base_length',␣
 ↪'base_anchors', 'base_params', 'base_queries', 'base_digits', 'base_entropy']
full_features = ['full_num_periods', 'full_spec_symbols', 'full_length',␣
 ↪'full_anchors', 'full_params', 'full_queries', 'full_digits', 'full_entropy']
```

[65]:
```
#df2_full = df2.drop(base_features,axis=1)
#df2_base = df2.drop(full_features,axis=1)


from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 100)
```

[ ]:
```

```

[66]:
```
from sklearn import svm

clf = svm.SVC(kernel='linear')
```

[67]:
```
SVM = svm.LinearSVC()
SVM.fit(X, y)
SVM.predict(X.iloc[:,:])
print(SVM.score(X,y))
```

```
0.8533834586466166
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\svm\base.py:929:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
```

[68]:
```
df2_full = df2.drop(base_features,axis=1)
df2_base = df2.drop(full_features,axis=1)
```

[69]:
```
#X, y = df2_full, df['flag']
```

14.1 Full URL SVM

[70]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_full,
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 100)
```

[71]:

```
from sklearn import svm

clf = svm.SVC(kernel='linear')
```

[72]:

```
SVM = svm.LinearSVC()
SVM.fit(X_full, y)
SVM.predict(X_full.iloc[:,:])
print(SVM.score(X_full,y))
```

```
0.6546723952738991
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\svm\base.py:929:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
```

14.2 Base URL SVM

[73]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_base,
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 100)
```

[74]:

```
SVM = svm.LinearSVC()
SVM.fit(X_full, y)
SVM.predict(X_full.iloc[:,:])
print(SVM.score(X_full,y))
```

```
C:\Users\sean\Anaconda3\lib\site-packages\sklearn\svm\base.py:929:
ConvergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
0.7271750805585392
```

15 MLP

[75]:

```
# make new df with urls dropped
drop_cols = ['url', 'flag', 'full_url', 'base_url','source', 'ip']

#urls_features = ['']

df2 = df.drop(drop_cols,axis=1)
print(df.columns)
#df2 = df[df.columns]
X, y = df2, df['flag']
```

```
Index(['url', 'flag', 'full_url', 'base_url', 'base_num_periods',
       'full_num_periods', 'base_spec_symbols', 'full_spec_symbols',
       'base_length', 'full_length', 'ip', 'full_anchors', 'base_anchors',
       'full_params', 'base_params', 'full_queries', 'base_queries',
       'full_digits', 'base_digits', 'full_entropy', 'base_entropy', 'source'],
      dtype='object')
```

[76]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 100)
```

[77]:

```
import sklearn as sk
from sklearn.neural_network import MLPClassifier

NN = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2),␣
 ↪random_state=1)
NN.fit(X, y)
NN.predict(X.iloc[:,:])
print(NN.score(X,y))
```

```
0.5026852846401718
```

[78]:

```
#df2_full = df2.drop(base_features,axis=1)
#df2_base = df2.drop(full_features,axis=1)
```

[79]:

```
#X_full, y = df2_full, df['flag']
```

[ ]:

```

```

[80]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_full,
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 100)
```

[81]:

```
import sklearn as sk
from sklearn.neural_network import MLPClassifier

NN_full = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(5, 2),␣
 ↪random_state=100)
NN_full.fit(X_full, y)
NN_full.predict(X_full.iloc[:,:])
print(NN_full.score(X_full, y))
```

```
0.5370569280343717
```

[82]:

```
#X_base, y = df2_base, df['flag']
```

15.1 Base URL MLP

[83]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_base,
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 100)
```

[84]:

```
import sklearn as sk
from sklearn.neural_network import MLPClassifier

NN_base = MLPClassifier(hidden_layer_sizes=(150,100,50),␣
 ↪max_iter=300,activation ='relu', solver='adam', random_state=100)
NN_base.fit(X_base, y)
NN_base.predict(X_base.iloc[:,:])
print(NN_base.score(X_base, y))
```

```
0.8469387755102041
```

15.2 Full URL MLP

[85]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_full,
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 100)
```

[86]:

```
import sklearn as sk
from sklearn.neural_network import MLPClassifier

NN_full = MLPClassifier(hidden_layer_sizes=(150,100,50),␣
 ↪max_iter=300,activation ='relu', solver='adam', random_state=100)
```

```
NN_full.fit(X_full, y)
NN_full.predict(X_full.iloc[:,:])
print(NN_full.score(X_full, y))
```

```
0.7953813104189044
```

15.3 Base URL Shannon Entropy MLP

[87]:

```
#X_ent = df[['base_entropy','full_entropy']]
X_ent = df[['base_entropy']]
```

[88]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_ent,
                                                    y,
                                                    test_size = 0.3,
                                                    random_state = 100)
```

[89]:

```
import sklearn as sk
from sklearn.neural_network import MLPClassifier

NN_ent = MLPClassifier(hidden_layer_sizes=(150,100,50), max_iter=300,activation␣
 ↪= 'relu',solver='adam',random_state=100)
NN_ent.fit(X_ent, y)
NN_ent.predict(X_ent.iloc[:,:])
print(NN_ent.score(X_ent,y))
```

```
0.7631578947368421
```

[ ]: