1.

2

* Check if use https and Issuer Is Trusted &and Age of Certificate≥ 1 Years or Using https and Issuer Is Not Trusted
  + If **YES**, check if % of URL Of Anchor <31%.
    - If **YES** then the website is LEGITIMATE
    - If **NO** then it also should be LEGITIMATE, but with lower confidence
  + If **NO**, check if % of URL Of Anchor <31%.
    - If **YES** then the website is LEGITIMATE
    - If **NO**, then website is PHISING

3.

The accuracy is 91%

4.

**from** **sklearn** **import** datasets, model\_selection, svm, metrics

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

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

**import** **threading**

**from** **sklearn.model\_selection** **import** train\_test\_split

**from** **sklearn** **import** tree

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

**from** **sklearn** **import** preprocessing **as** pp

*##Prepare data##*

filename=r'phishing.csv'

data\_train=pd.read\_csv(filename,index\_col=**None**,na\_values='?',sep = ';')

data\_train=data\_train.dropna()

colnames = data\_train.columns.get\_values()

print("**\n**DESCRIBE DATA:**\n**",data\_train.describe())

data\_train.shape

DESCRIBE DATA:

having\_IP\_Address URL\_Length Shortining\_Service having\_At\_Symbol \

count 11055.000000 11055.000000 11055.000000 11055.000000

mean 0.313795 -0.633198 0.738761 0.700588

std 0.949534 0.766095 0.673998 0.713598

min -1.000000 -1.000000 -1.000000 -1.000000

25% -1.000000 -1.000000 1.000000 1.000000

50% 1.000000 -1.000000 1.000000 1.000000

75% 1.000000 -1.000000 1.000000 1.000000

max 1.000000 1.000000 1.000000 1.000000

double\_slash\_redirecting Prefix\_Suffix having\_Sub\_Domain \

count 11055.000000 11055.000000 11055.000000

mean 0.741474 -0.734962 0.063953

std 0.671011 0.678139 0.817518

min -1.000000 -1.000000 -1.000000

25% 1.000000 -1.000000 -1.000000

50% 1.000000 -1.000000 0.000000

75% 1.000000 -1.000000 1.000000

max 1.000000 1.000000 1.000000

SSLfinal\_State Domain\_registeration\_length Favicon ... \

count 11055.000000 11055.000000 11055.000000 ...

mean 0.250927 -0.336771 0.628584 ...

std 0.911892 0.941629 0.777777 ...

min -1.000000 -1.000000 -1.000000 ...

25% -1.000000 -1.000000 1.000000 ...

50% 1.000000 -1.000000 1.000000 ...

75% 1.000000 1.000000 1.000000 ...

max 1.000000 1.000000 1.000000 ...

popUpWindow Iframe age\_of\_domain DNSRecord web\_traffic \

count 11055.000000 11055.000000 11055.000000 11055.000000 11055.000000

mean 0.613388 0.816915 0.061239 0.377114 0.287291

std 0.789818 0.576784 0.998168 0.926209 0.827733

min -1.000000 -1.000000 -1.000000 -1.000000 -1.000000

25% 1.000000 1.000000 -1.000000 -1.000000 0.000000

50% 1.000000 1.000000 1.000000 1.000000 1.000000

75% 1.000000 1.000000 1.000000 1.000000 1.000000

max 1.000000 1.000000 1.000000 1.000000 1.000000

Page\_Rank Google\_Index Links\_pointing\_to\_pageStatistical\_report \

count 11055.000000 11055.000000 11055.000000 11055.000000

mean -0.483673 0.721574 0.344007 0.719584

std 0.875289 0.692369 0.569944 0.694437

min -1.000000 -1.000000 -1.000000 -1.000000

25% -1.000000 1.000000 0.000000 1.000000

50% -1.000000 1.000000 0.000000 1.000000

75% 1.000000 1.000000 1.000000 1.000000

max 1.000000 1.000000 1.000000 1.000000

Result

count 11055.000000

mean 0.113885

std 0.993539

min -1.000000

25% -1.000000

50% 1.000000

75% 1.000000

max 1.000000

[8 rows x 31 columns]

(11055, 31)

**from** **sklearn.tree** **import** DecisionTreeClassifier

*###MAKE DECISION TREE###*

*#data\_train*

X\_all = data\_train.drop(['Result'], axis=1)

y\_all = data\_train['Result']

test\_size=2050

train\_size=8050

X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X\_all, y\_all, test\_size=test\_size, train\_size=train\_size)

### Notes

On default settings on tree makes bit unbalanced classification. fine tuned min\_samples\_split value to 1500. min\_samples\_leaf value tuning didn't really do nothing special between range 2-100 but if it is increased to 200 balance of the classification gets worse.

**from** **sklearn.metrics** **import** confusion\_matrix

**from** **sklearn.metrics** **import** precision\_score,confusion\_matrix,f1\_score,classification\_report

clf = DecisionTreeClassifier(criterion="gini",

min\_samples\_split=1500,

min\_samples\_leaf=50,

splitter='best',

max\_depth=2)

clf.fit(X\_train, y\_train)

pre = clf.predict(X\_test)

accuracy\_score = metrics.accuracy\_score(y\_test, pre)

print("accuracy:", accuracy\_score)

cf\_matrix = confusion\_matrix(y\_test, pre)

print()

print(pd.crosstab(y\_test, pre, rownames=['True'], colnames=['Predicted'], margins=**True**),"**\n**")

print(classification\_report(y\_test,pre))

accuracy: 0.9102439024390244

Predicted -1 1 All

True

-1 809 85 894

1 99 1057 1156

All 908 1142 2050

precision recall f1-score support

-1 0.89 0.90 0.90 894

1 0.93 0.91 0.92 1156

accuracy 0.91 2050

macro avg 0.91 0.91 0.91 2050

weighted avg 0.91 0.91 0.91 2050

**from** **sklearn.tree** **import** plot\_tree

**import** **graphviz**

plt.figure()

plot\_tree(clf, filled=**True**,class\_names = ['legitimate','phising'])

plt.show()

**from** **sklearn.tree** **import** export\_graphviz

*# Export as dot file*

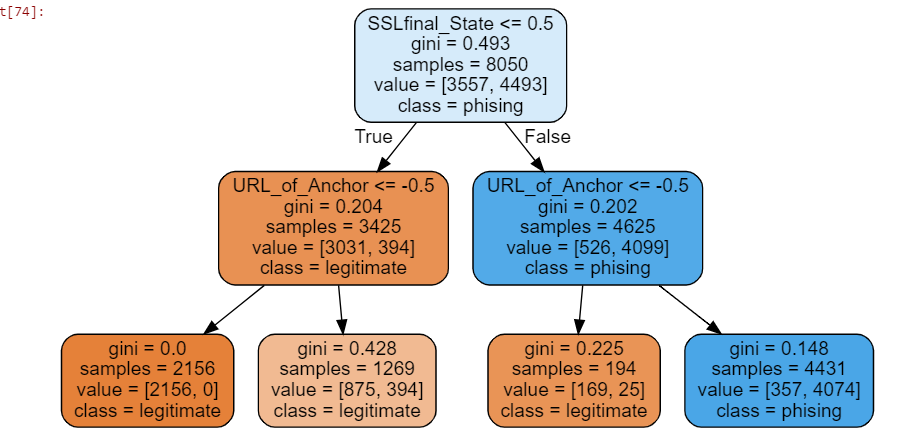
export\_graphviz(clf, out\_file='tree.dot',

feature\_names = colnames[:30],

class\_names = ['legitimate','phising'],

rounded = **True**, proportion = **False**,

precision = 2, filled = **True**)



*##Edited plot*

**import** **pydotplus**

**import** **pydot**

**import** **graphviz**

*#Without this graphviz exetables not found*

**import** **os**

os.environ['PATH'] = os.environ['PATH']+';'+os.environ['CONDA\_PREFIX']+r"\Library\bin\graphviz"

tree\_data = export\_graphviz(clf, feature\_names=colnames[:30], out\_file=**None**,class\_names = ['legitimate','phising'], filled=**True**, rounded=**True**)

tree\_graph = pydotplus.graph\_from\_dot\_data(tree\_data)

tree\_graph.set\_size('"10,10!"')

tree\_graph.write\_png('img\_treeDot.png')

*#tree\_graph.write\_pdf("pdf\_treeDot.pdf")*

graphviz\_graph = graphviz.Source(tree\_graph.to\_string())

graphviz\_graph

<graphviz.files.Source at 0x1ff5f82c808>

Plot is on the top of the document