Malicious URL Prediction

411021314 林芝蓁

I. Objective

This project aims to use the big data analyze method to predict if an URL is malicious or not.

II. Tools

In this project, we mainly use Python and the following modules: requests, os, shutil, csv, apscheduler, pandas, seaborn, mypyplot and sklearn, etc.

III. Dataset Description

A. Source

The dataset—ISCX-URL2016 comes from Canadian Institute for Cybersecurity (CIC). Here is the link: https://www.unb.ca/cic/datasets/url-2016.html

B. Download Method

We wrote a program to update the dataset daily:

fetch_data_scheduler.py > ...

```python
def fetch_data():
    # remove the current file
    print("fetch start")
    import os
    import shutil
    if os.path.exists(".\ISCXURL2016.zip"):
        os.remove(".\ISCXURL2016.zip")
    if os.path.exists(".\ISCXURL2016"):
        shutil.rmtree(".\ISCXURL2016")
    #  download a file from the web using requests
    import requests
    url = 'http://205.174.165.80/CICDataset/ISCX-URL-2016/Dataset/ISCXURL2016.zip'
    r = requests.get( url , allow_redirects=True )
    open('ISCXURL2016.zip','wb').write(r.content)
    from zipfile import ZipFile
    with ZipFile('ISCXURL2016.zip', 'r') as zipObj:
        # Extract all the contents of zip file in different directory
        zipObj.extractall('ISCXURL2016')
        # print('File is unzipped in dataset folder')
    print("Data_fetched")

from apscheduler.schedulers.blocking import BlockingScheduler
scheduler = BlockingScheduler()
# scheduler.add_job(fetch_data, 'interval', days=1)
scheduler.add_job(fetch_data, 'interval', seconds=5)
scheduler.start()
```

C. Fields

There are 36708 rows and each has 80 columns:

36698	0	4	5	6.5	12	
36699	941	3	6	4.333334	9	
36700	51	3	12	6.666667	16	
36701	6	3	7	2.666667	4	
36702	0	3	6	2.666667	4	
36703	0	3	5	4.666667	10	
36704	29	4	14	5.75	12	3.6
36705	0	4	13	3.75	8	8.4
36706	58	3	27	6.666667	16	
36707	35	3	13	4.333334	9	
36708	40	3	25	6.666667	16	
36709						

	BX	BY	BZ	CA	CB
y_D	Entropy_D	Entropy_F	Entropy_E	Entropy_A	URL_Type_obf_Type
493	0.894886	0.850608	NaN	-1	Defacement
493	0.814725	0.859793	0	-1	Defacement
493	0.814725	0.80188	0	-1	Defacement
493	0.814725	0.66321	0	-1	Defacement
493	0.814725	0.804526	0	-1	Defacement
493	0.814725	0.755658	0	-1	Defacement
493	0.814725	0.766719	0	-1	Defacement
493	0.814725	0.797498	0	-1	Defacement
493	0.814725	0.732258	0	-1	Defacement
493	0.894886	0.894886	NaN	-1	Defacement
493	0.810169	0.804	0	-1	Defacement

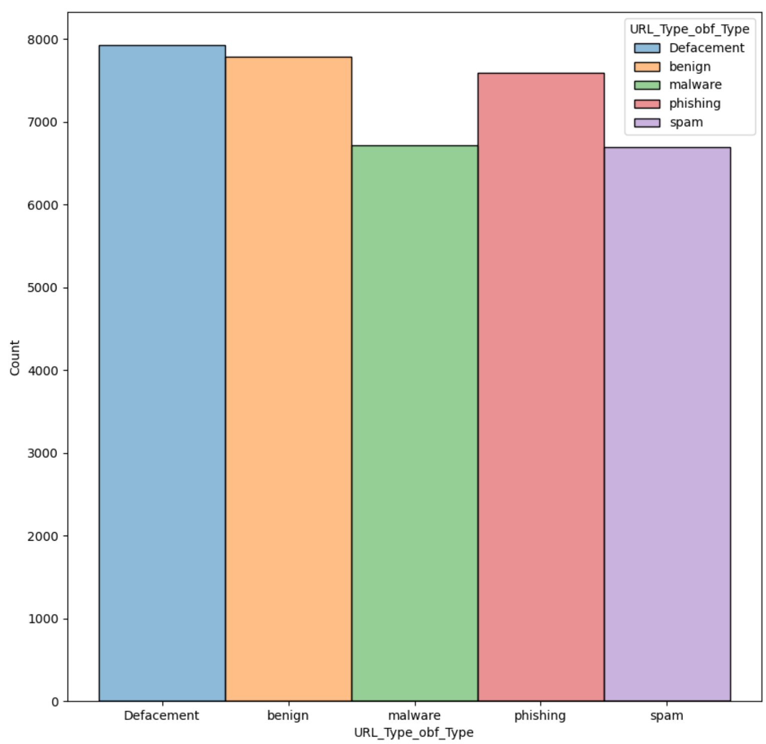
Each row represents one URL and its columns are its attributes, e.g., Domain Length, Token count of path, Entropy or number of dots, etc.

For the last column, it's the URL type of the corresponding URL. The 5 types are "Dafacement", "Benign", "Malware", "Phishing", "Spam", except the "Benign" the other 4 types are unreliable.

D. Analyze of the dataset

Here's the number of each of the 5 types and the histogram:

	A	B	C	D	E
1	Defaceme	Benign	Malware	Phishing	Spam
2	7930	7781	6712	7586	6698

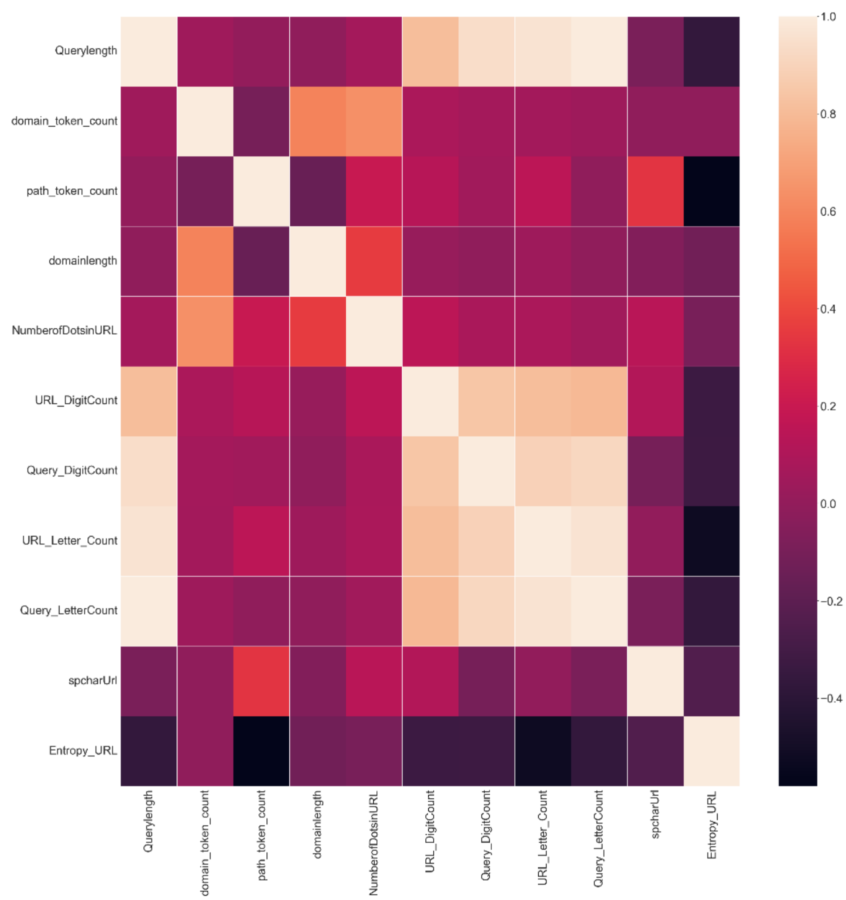


IV. Feature Selection

To select the feature, we first consider the lexical properties of an URL, like length, Token Count, Special characters count, etc. First we have 12 feature candidates. To reduce the parameters of the training model, we calculate the correlation of the 12 features and only remain 1 features among those highly correalated.

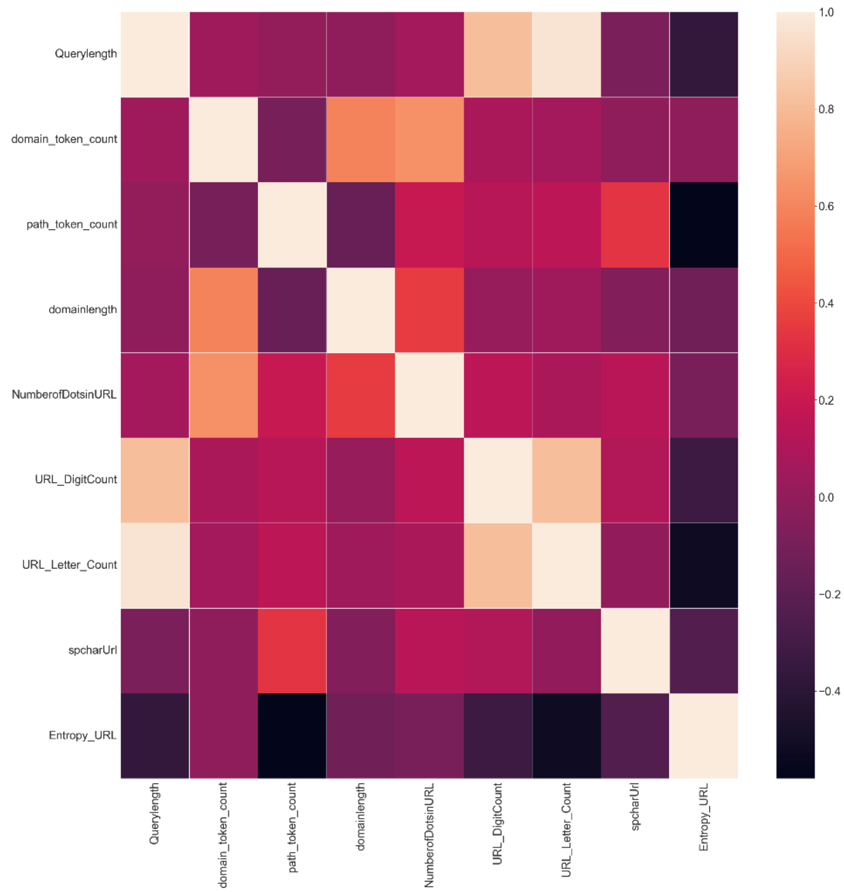
	A	B	C	D	E	F	G	H	I	J	K	L	M
1		Querylength	domain_token_count	path_token_count	domainlength	pathLength	NumberofDotsinURL	URL_DigitCount	Query_DigitCount	URL_Letter_Count	Query_LetterCount	spcharUrl	Entropy_URL
2	Querylength	1	0.045174451	0.004259119	-0.006624583	0.957952734	0.062250638	0.813093062	0.94104856	0.967339102	0.997635785	-0.09228	-0.369418161
3	domain_token_count	0.045174451	1	-0.101013481	0.591224559	0.026922616	0.641196773	0.090232748	0.062196534	0.059136256	0.04065333	-0.00752	-0.012042547
4	path_token_count	0.004259119	-0.101013481	1	-0.159773894	0.217111465	0.196373809	0.131080122	0.050160021	0.152752066	-0.011120188	0.32837	-0.580159976
5	domainlength	-0.006624583	0.591224559	-0.159773894	1	-0.042816436	0.35760835	0.016246223	-0.011757183	0.038210988	-0.007584475	-0.05689	-0.120619318
6	pathLength	0.957952734	0.026922616	0.217111465	-0.042816436	1	0.092365348	0.867555968	0.905751823	0.988851755	0.954079897	0.03416	-0.515836559
7	NumberofDotsinURL	0.062250638	0.641196773	0.196373809	0.35760835	0.092365348	1	0.150042244	0.081583393	0.089918086	0.052575872	0.13856	-0.097068082
8	URL_DigitCount	0.813093062	0.090232748	0.131080122	0.016246223	0.867555968	0.150042244	1	0.845644579	0.810567868	0.794385274	0.11295	-0.330339706
9	Query_DigitCount	0.94104856	0.062196534	0.050160021	-0.011757183	0.905751823	0.081583393	0.845644579	1	0.892292421	0.916617178	-0.10252	-0.326519356
10	URL_Letter_Count	0.967339102	0.059136256	0.152752066	0.038210988	0.988851755	0.089918086	0.810567868	0.892292421	1	0.968207502	-0.00021	-0.519121109
11	Query_LetterCount	0.997635785	0.04065333	-0.011120188	-0.007584475	0.954079897	0.052575872	0.794385274	0.916617178	0.968207502	1	-0.08984	-0.367569282
12	spcharUrl	-0.092284737	-0.007524432	0.328365287	-0.056886104	0.034156313	0.138557239	0.112949661	-0.102523269	-0.000214539	-0.089843252	1	-0.245282415
13	Entropy_URL	-0.369418161	-0.012042547	-0.580159976	-0.120619318	-0.515836559	-0.097068082	-0.330339706	-0.326519356	-0.519121109	-0.367569282	-0.24528	1

The highlighted ones are all above 0.9.



After discarding the highly correlated features, there are 9 features remain:

	A	B	C	D	E	F	G	H	I	J
1		Querylength	domain_token_count	path_token_count	domainlength	NumberofDotsinURL	URL_DigitCount	URL_Letter_Count	spcharUrl	Entropy_URL
2	Querylength	1	0.045174451	0.004259119	-0.006624583	0.062250638	0.813093062	0.967339102	-0.092284737	-0.369418161
3	domain_token_count	0.045174451	1	-0.101013481	0.591224559	0.641196773	0.090232748	0.059136256	-0.007524432	-0.012042547
4	path_token_count	0.004259119	-0.101013481	1	-0.159773894	0.196373809	0.131080122	0.152752066	0.328365287	-0.580159976
5	domainlength	-0.006624583	0.591224559	-0.159773894	1	0.35760835	0.016246223	0.038210988	-0.056886104	-0.120619318
6	NumberofDotsinURL	0.062250638	0.641196773	0.196373809	0.35760835	1	0.150042244	0.089918086	0.138557239	-0.097068082
7	URL_DigitCount	0.813093062	0.090232748	0.131080122	0.016246223	0.150042244	1	0.810567868	0.112949661	-0.330339706
8	URL_Letter_Count	0.967339102	0.059136256	0.152752066	0.038210988	0.089918086	0.810567868	1	-0.000214539	-0.519121109
9	spcharUrl	-0.092284737	-0.007524432	0.328365287	-0.056886104	0.138557239	0.112949661	-0.000214539	1	-0.245282415
10	Entropy_URL	-0.369418161	-0.012042547	-0.580159976	-0.120619318	-0.097068082	-0.330339706	-0.519121109	-0.245282415	1



The reason why we keep the feature "querylength" is that one URL is either has query or not. To distinguish them, we keep this feature.

V. Analyze Method

For analyzing the dataset and prediction, we use the decision tree to classify each URL to the 5 URL type classes. And the module used is sklearn.

Here's the training function:

```python
# 1_training.py > ...
import pandas
import numpy
pandas.options.mode.chained_assignment = None
'''See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  features.replace([numpy.inf, -numpy.inf], numpy.nan, inplace=True)
C:\AMP\Apache24\htdocs\Big Data Final Project\1_training.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame'''
DATASET = pandas.read_csv(".\ISCXURL2016\FinalDataset\All.csv")
def training( train_features , ratio_of_train = 0.3 ): # selected_features: [ 0 , 1 , 2 , 20 , 21 , 34 , 38 , 43 , 44 , 49 , 57 , 73 ]
    features = DATASET.iloc[ : , train_features ] # X
    features.replace([numpy.inf, -numpy.inf], numpy.nan, inplace=True)
    features.dropna()
    url_type = DATASET.iloc[ : , 79 : 80 ] # Y

    from sklearn.model_selection import train_test_split
    from sklearn import tree
    # train_test_split( X , Y , test_size )
    x_train , x_test , y_train , y_test = train_test_split( features , url_type , test_size = ratio_of_train )
    ### print("Train Data")
    ### print( x_train )
    ### print( y_train )
    DT = tree.DecisionTreeClassifier()
    res = DT.fit( x_train , y_train )
    return res , x_test , y_test , len( url_type )
```

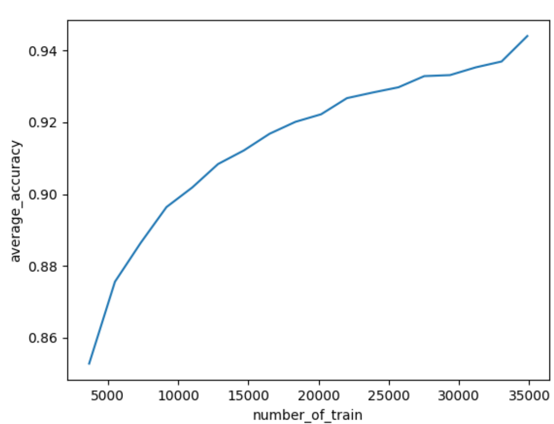
VI. Evaluation

For evaluating the accuracy of the trained model, we use the cross-validation to prevent the overfitting problem and we take the average accuracy for more precise evaluation.

```python
# 1_test_accuracy.py > ...
import pandas
training = __import__('1_training')

def test_Accuracy( ratio_of_train , train_features , training_times ):
    total_accuracy = 0
    # train the modal for n times
    for k in range( 0 , training_times ):
        # get the trained modal and x_test, y_test
        res , x_test , y_test , N = training.training( train_features , ratio_of_train ) # N == len( x_train ) + len( x_test )
        prediction = res.predict( x_test )
        y_test = ( pandas.DataFrame( y_test ) ).values.tolist()
        # print( x_test )
        # calculate the accuracy of current trained modal
        misPrediction = 0
        for i in range( 0 , len( prediction ) ):
            if( prediction[ i ] != y_test[ i ][ 0 ] ):
                misPrediction = misPrediction + 1
        # print( "misPrediction:" + str( misPrediction ) )
        accuracy = ( len( prediction ) - misPrediction ) / len( prediction )
        # print( "Accuracy: " + str( k ) + " " + str( accuracy ) )
        total_accuracy = total_accuracy + accuracy
    return total_accuracy / training_times , N - len( x_test )
```

Following is the accuracy with the varying size of training data and the code:

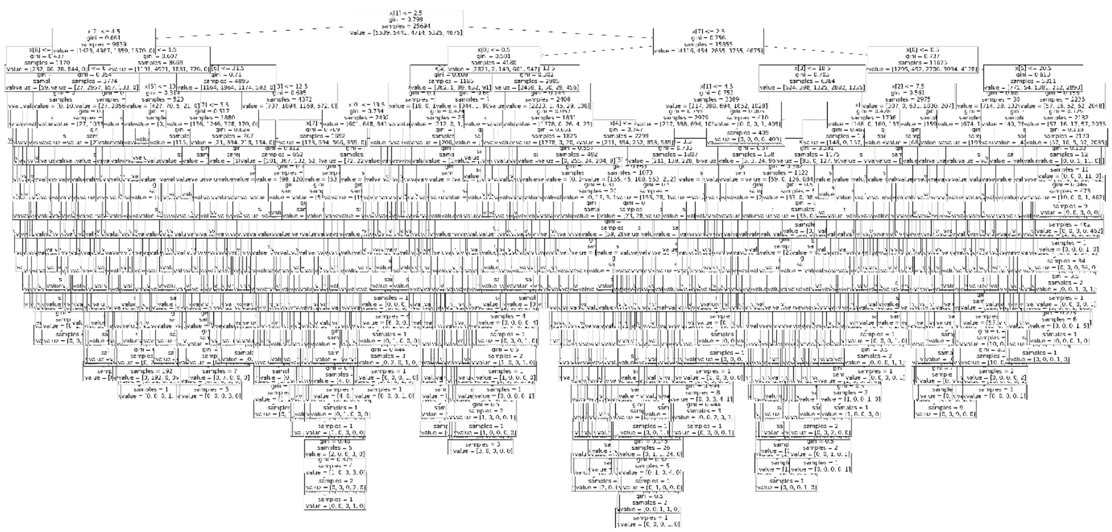
	A	B	C	D
1		test_size	average_accuracy	number_of_train
2	0	0.9	0.854729546	3670
3	1	0.85	0.875648216	5506
4	2	0.8	0.88642648	7341
5	3	0.75	0.894689623	9176
6	4	0.7	0.902981125	11012
7	5	0.65	0.907359598	12847
8	6	0.6	0.912267877	14682
9	7	0.55	0.916469365	16518
10	8	0.5	0.920393375	18353
11	9	0.45	0.922840366	20188
12	10	0.4	0.927228768	22024
13	11	0.35	0.927646326	23859
14	12	0.3	0.929901026	25694
15	13	0.25	0.931110385	27530
16	14	0.2	0.935630618	29365
17	15	0.15	0.935917923	31200
18	16	0.1	0.936257151	33036
19	17	0.05	0.936220044	34871



```python
######################################################## csv for varying number of test data ####################################################################
accuracy_csv = pandas.DataFrame()
test_size_list = [ 0.9 , 0.85 , 0.8 , 0.75 , 0.7 , 0.65 , 0.6 , 0.55 , 0.5 , 0.45 , 0.4 , 0.35 , 0.3 , 0.25 , 0.2 , 0.15 , 0.1 , 0.05 ]
accuracy_csv['test_size'] = test_size_list
avg_acc = []
num_train_list = []
# print( accuracy_csv )
NUM_TRAINING = 10
test_size = 0.9 # percentage of dataset used as test data
while( test_size > 0 ):
    cur_avg_acc , cur_num_train = test_Accuracy( ratio_of_train=test_size , train_features=[ 0 , 1 , 2 , 20 , 34 , 38 , 44 , 57 , 73 ] , training_times = NUM_TRAINING )
    avg_acc.append( cur_avg_acc )
    num_train_list.append( cur_num_train )
    # print("Average Accuracy for " + str( NUM_TRAINING ) + " times training with test size " + str( test_size ) + ": " + str( cur_avg_acc ) )
    test_size = test_size - 0.05
accuracy_csv['average_accuracy'] = avg_acc
accuracy_csv['number_of_train'] = num_train_list
accuracy_csv.to_csv('.\\1_accuracy\\average_accuracy_vary_test_size.csv')

###################################################### lineplot for varying number of test data ##################################################################
import matplotlib.pyplot as plt
import seaborn
plt.clf()
# print( accuracy_csv )
seaborn.lineplot( accuracy_csv , x='number_of_train' , y='average_accuracy')
plt.savefig('.\\1_accuracy\\average_accuracy_vary_test_size.png')
```

We can see the accuracy is above 85% overall, and the highest accuracy can be up to 93%.

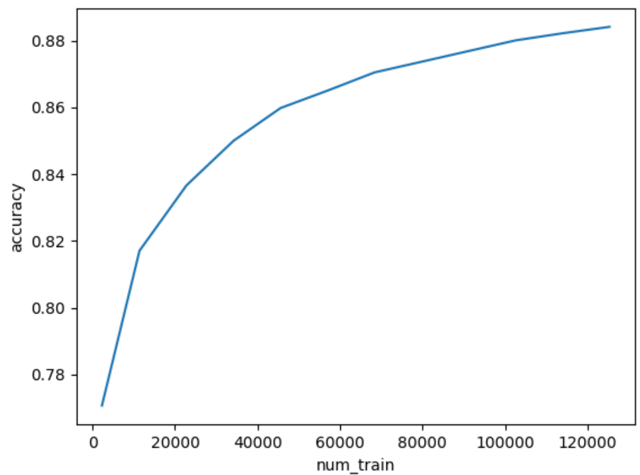
However the tree is actually complicated, for the reduction of the trained model, I'll present the reduced tree in the section VIII.



VII. Application

To achieve the URL prediction, we wrote a program to extract the feature from another URL dataset: https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset

The accuracy can be up to 88%:



Finally, **<u>we use the trained module to predict the URL entered by users:</u>**

```python
import validators
import pandas
training_with_url = __import__('2_training_with_url')
extract_url_features = __import__('2_extract_url_features')
# predict with All-URL-trained modal
# res = training_with_url.get_trained_modal( test_size = 0.1 )

# predict with Kaggle-URL-trained modal
res = training_with_url.get_trained_modal_Kaggle()

while( 1 ):
    url = input("Enter an URL:")
    if( url == 'exit' ):
        break
    if( validators.url( url ) ):
        # print("Valid")
        feature = extract_url_features.feature( url )
        feature_list = feature.getFeatureList()
        feature_df = pandas.DataFrame(columns=['Querylength', 'domain_token_count' , 'path_token_count' , 'domainlength', 'pathLength', 'NumberofDotsinURL', 'URL_DigitCount', 'URL_
        feature_df = pandas.concat( [ feature_df , pandas.DataFrame( columns=['Querylength', 'domain_token_count' , 'path_token_count' , 'domainlength', 'pathLength', 'NumberofDots
        prediction = res.predict( feature_df )
        print( feature_df )
        print(prediction)
        print("prediction:" + prediction[ 0 ] )
    else:
        print("Invalid URL !")
```

```
Enter an URL:https://www.unb.ca/cic/datasets/url-2016.html
  Querylength domain_token_count path_token_count domainlength pathLength NumberofDotsinURL URL_DigitCount URL_Letter_Count spcharUrl
0           0                  3                5           10         26                 3              4               31         2
['malware']
prediction:malware
Enter an URL:Traceback (most recent call last):
  File "C:\AMP\Apache24\htdocs\Big Data Final Project\2_predict.py", line 12, in <module>
    url = input("Enter an URL:")
          ^^^^^^^^^^^^^^^^^^^^^^
KeyboardInterrupt
PS C:\AMP\Apache24\htdocs\Big Data Final Project> python 2_predict.py
Enter an URL:https://www.unb.ca/cic/datasets/url-2016.html
  Querylength domain_token_count path_token_count domainlength pathLength NumberofDotsinURL URL_DigitCount URL_Letter_Count spcharUrl
0           0                  3                5           10         26                 3              4               31         2
['benign']
prediction:benign
Enter an URL:
```

VIII. Modification After Presentation

Before the presentation, we didn't limit the maximum depth of the tree, also, the number of features still had the chance to be reduced to decrease the complexity of Tree. Hence, we wrote a program to take the minimum acceptable depth of the decision tree and reduce the selected features:

```python
def test_depth_acc( limit = 15 ):
    d = []
    acc = []
    features = [ 0 , 1 , 2 , 20 , 34 , 38 , 44 , 57 , 73 ]
    features_names = ['Query Length' , 'Domain Token Count' , 'Path Token Count' , 'Domain Length' , 'Entropy' , 'URL Digit Count' , 'URL Letter Count' , 'Number of Special Charact
    for i in range( 1 , limit + 1 ):
        plotTree( features , 0.3 , i , feature_names=features_names )
        ac , s = test_Accuracy( 0.3 , features , 10 , i )
        d.append( i )
        acc.append( ac )
        print( "max_depth:" + str( i ) + " acc:" +  str( ac ) )
    df_ = pandas.DataFrame()
    df_['max_depth'] = d
    df_['accuracy'] = acc
    plt.clf()
    seaborn.lineplot( df_ , x='max_depth' , y='accuracy' )
    plt.savefig('.\\5\\depth_acc.png')
    df_.to_csv('.\\5\\depth_acc.csv' , index=None )

def test_reduce_features( features , features_names , max_depth ):
    acc = []
    remove = []
    for i in range( 0 , len( features ) ):
        tmp_features = features.copy()
        tmp_features.remove( tmp_features[ i ] )
        ac , s = test_Accuracy( 0.3 , tmp_features , 10 , max_depth=max_depth )
        acc.append( ac )
        remove.append( features_names[ i ] )
    df_ = pandas.DataFrame()
    df_['removed_feature'] = remove
    df_['accuracy'] = acc
    plt.clf()
    plt.figure(figsize=(20 , 20))
    seaborn.lineplot( df_ , x='removed_feature' , y='accuracy' )
    plt.savefig('.\\5\\' + str( i_th ) + '_th_remove_acc')
    return acc.index( max( acc ) ) , max( acc ) # return the index of the max accuracy
```

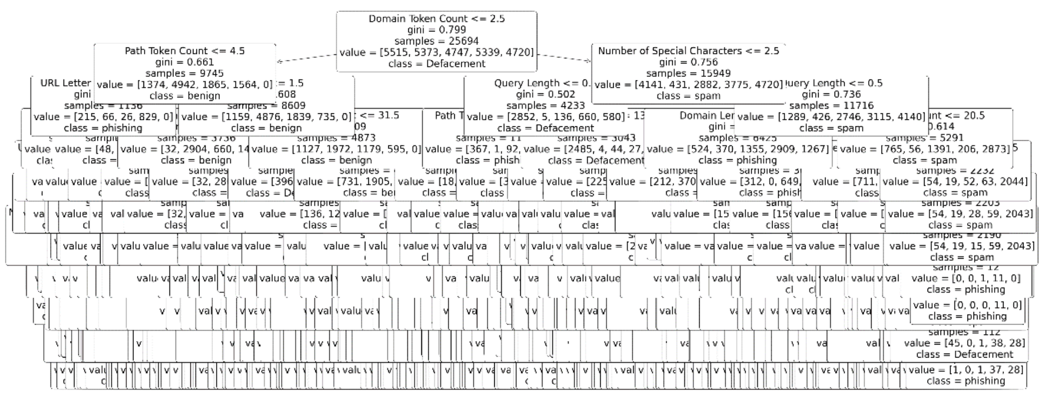
```python

#######################################################################################################################################################
#######################################################################################################################################################
#######################################################################################################################################################
#######################################################################################################################################################
#################################################### test the acceptable depth for the tree ##########################################################
#######################################################################################################################################################
#######################################################################################################################################################
#######################################################################################################################################################
#######################################################################################################################################################
test_depth_acc( limit = 10 )



#######################################################################################################################################################
#######################################################################################################################################################
#######################################################################################################################################################
#######################################################################################################################################################
#######################################################################################################################################################
######################################################### reduce the number of features ##############################################################
#######################################################################################################################################################
#######################################################################################################################################################
#######################################################################################################################################################
#######################################################################################################################################################
#######################################################################################################################################################
# [ 0 , 1 , 2 , 20 , 34 , 38 , 44 , 57 , 73 ]
i_th = 0
features = [ 0 , 1 , 2 , 20 , 34 , 38 , 44 , 57 , 73 ]
features_names = ['Query Length' , 'Domain Token Count' , 'Path Token Count' , 'Domain Length' , 'Entropy' , 'URL Digit Count' , 'URL Letter Count' , 'Number of Special Characters'
max_acc = 0.99
while( len( features ) > 0 and max_acc > 0.85 ):
    next_remove_feature_index , max_acc = test_reduce_features( features_names=features_names , features=features , max_depth=11 )
    print( "next_remove:" + features_names[ next_remove_feature_index ] )
    i_th = i_th + 1
    features.remove( features[ next_remove_feature_index ] )
    features_names.remove( features_names[ next_remove_feature_index ] )
```

There are two functions called ***test_depth_acc*** and ***test_reduce_features*** with keeping increase the depth of the tree and see how many depth we need to achieve the acceptable accuracy and keeping reduce the features for deleting which we get the maximum accuracy. Finally we called the two functions to get the following result:

A. Find the tree depth with the acceptable accuracy with the 9 features

	A	B	C
1	max_depth	accuracy	
2	1	0.376918	
3	2	0.493017	
4	3	0.573876	
5	4	0.62182	
6	5	0.673577	
7	6	0.722219	
8	7	0.771388	
9	8	0.815227	
10	9	0.843312	
11	10	0.871143	

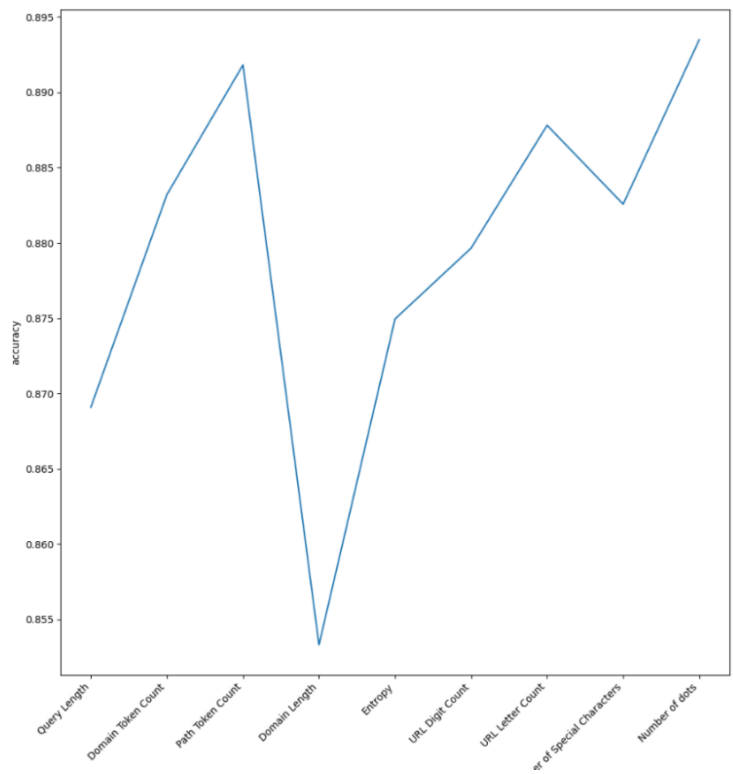
We can see that **to achieve the accuracy above 85%, we need at least 10 layers with the 9 features**. Still, the tree looks extremely complicated, however, it's much better than the previous one with the unlimited depth.



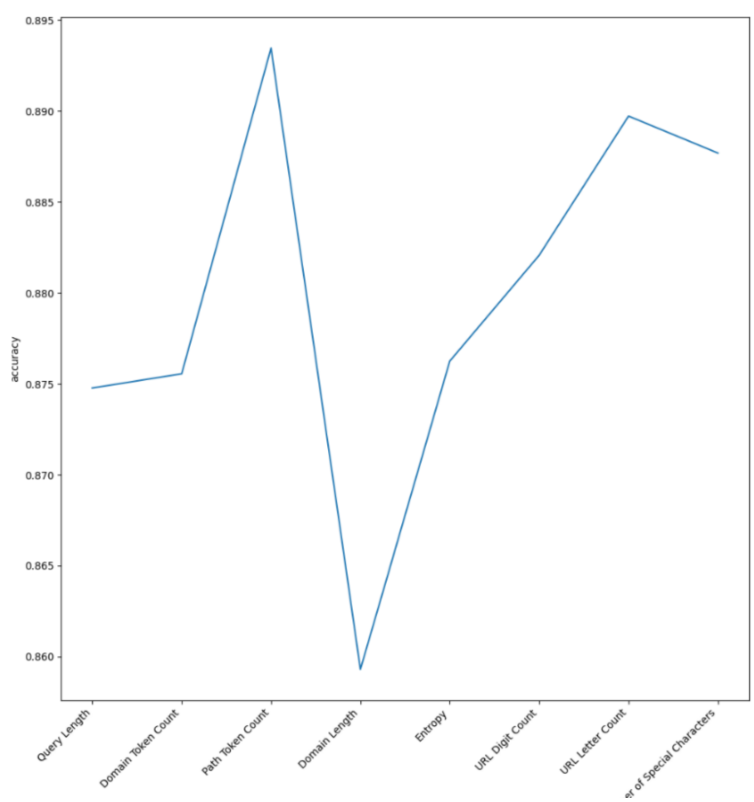
B. Reduce the number of features with no limit max depth

The following shows the process of finding the discarded features with the highest accuracy and the line plots(x axis is the discarded features):

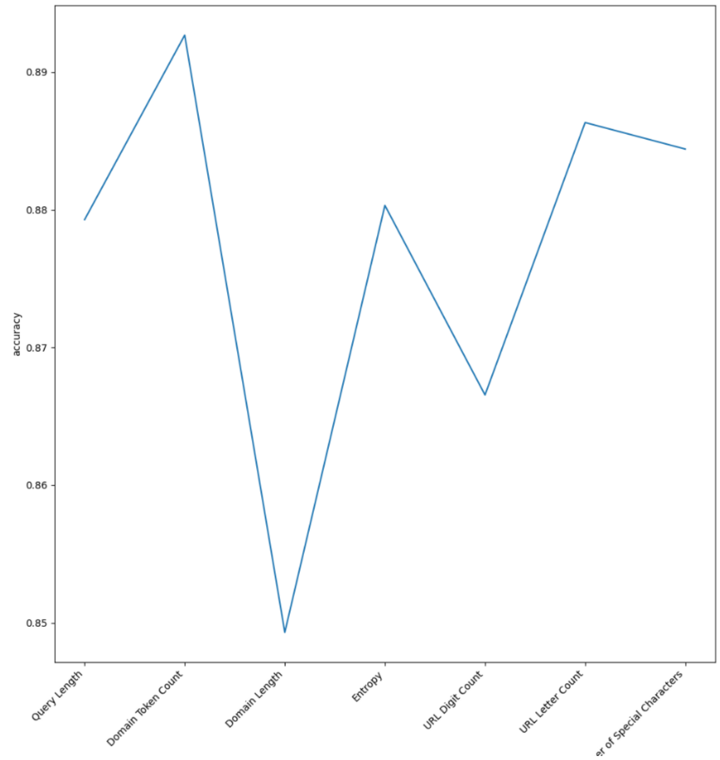
1. Originally we have 9 features, and we are going to test the accuracy with discarding each features of the remaining features, then take the combination of features with highest accuracy.
2. First, we discard the feature "number of dots



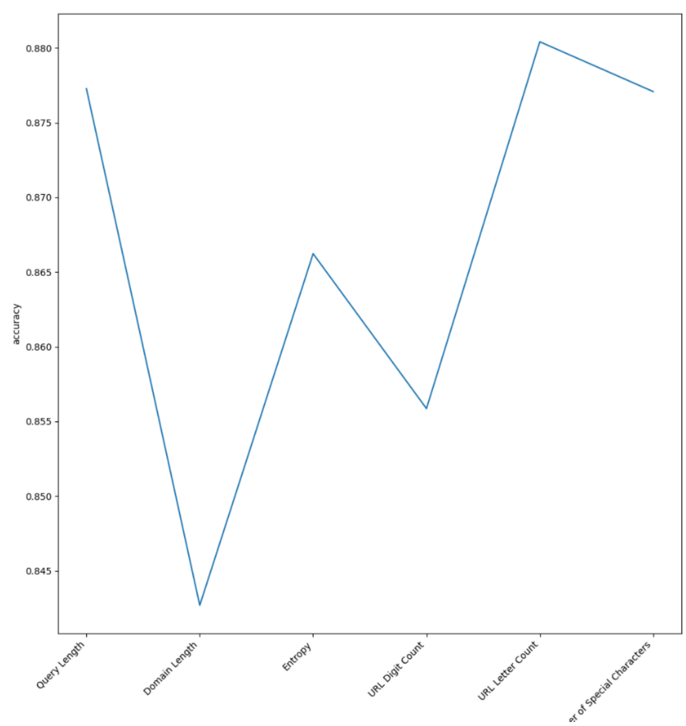
3. Next we discard the feature “Path Token Count”



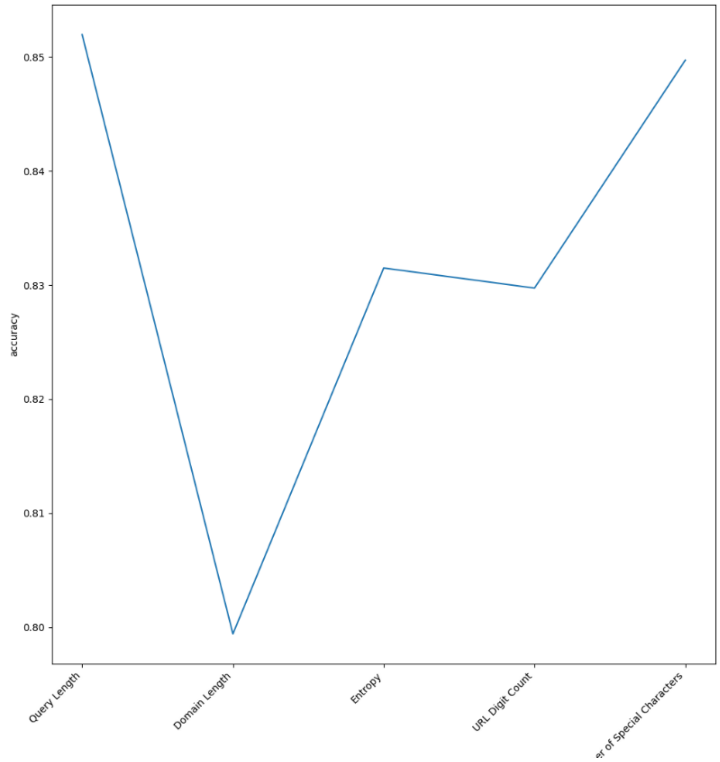
4. Then discard the feature “Domain Token Count”



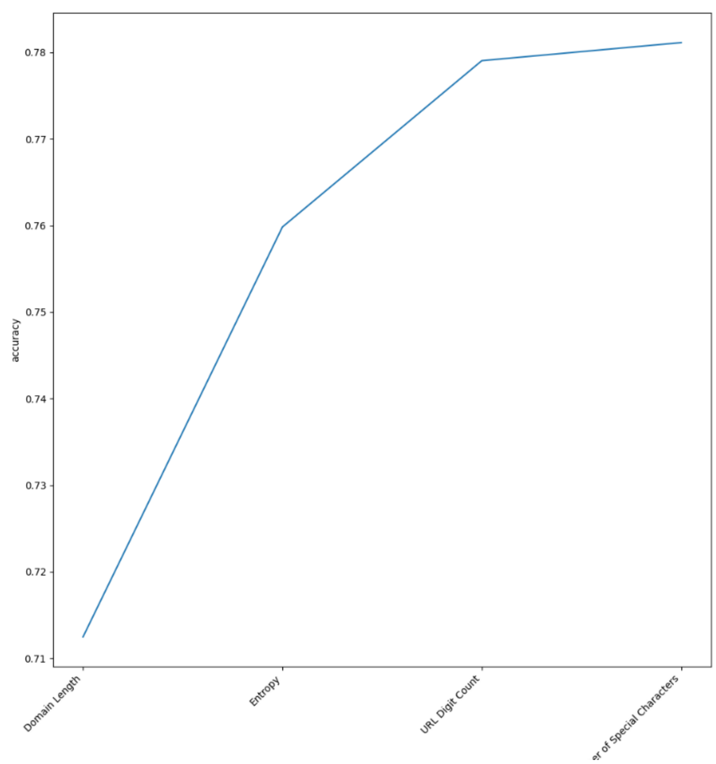
5. Next, discard the feature “Letter Count”



6. Next, discard the feature “Query Length”



7. Finally, we find that we cannot reduce the number of features anymore since the accuracy will be below 85%, actually it’s even



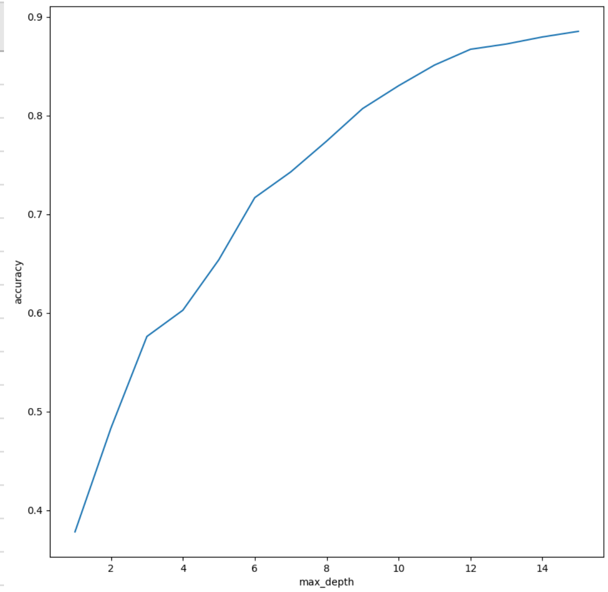
below 80%.

The remaining features are Domain Length, Entropy, URL Digit Count, and Number of Special Characters.

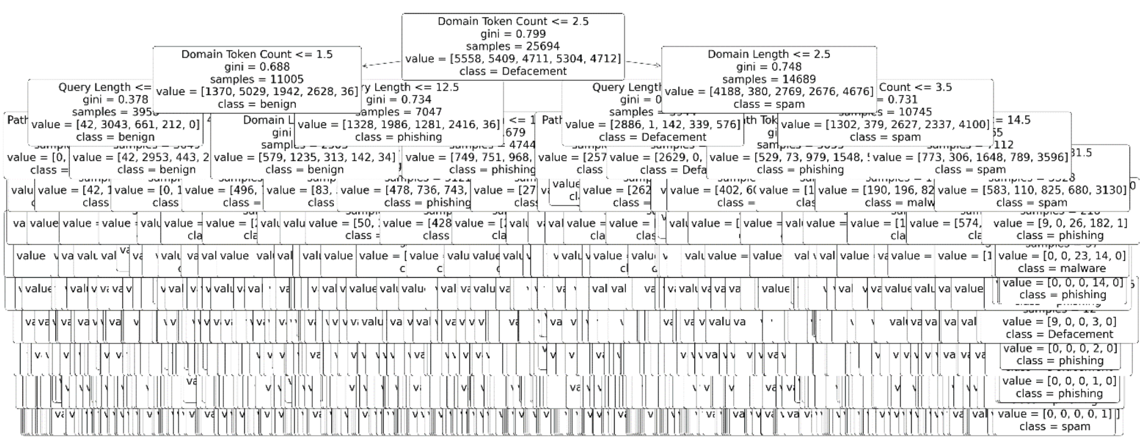
C. <u>Find the minimum depth of tree with the remaining 4 features</u>

We use the remaining 4 feature in the previous point B to test the minimum depth with the acceptable accuracy above 85%, then we get the result:

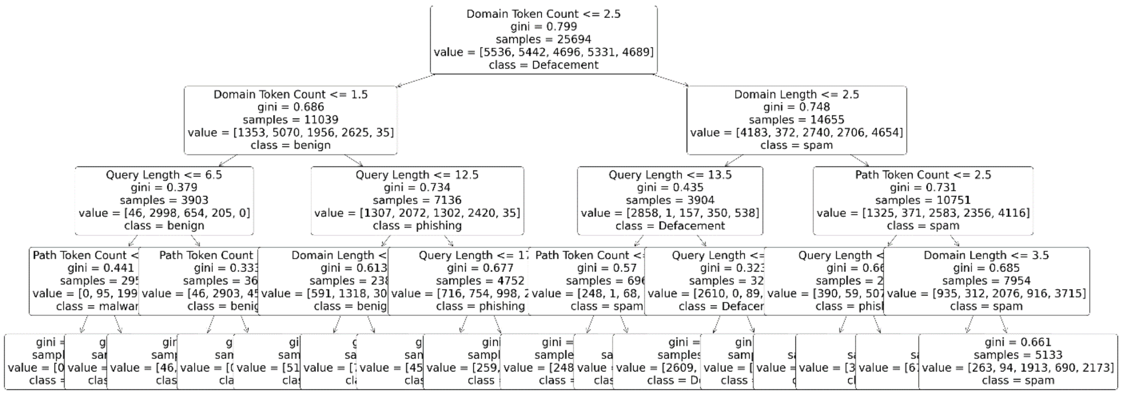
	A	B
1	max_depth	accuracy
2	1	0.378144
3	2	0.483256
4	3	0.576046
5	4	0.602642
6	5	0.653746
7	6	0.716698
8	7	0.742722
9	8	0.774003
10	9	0.806919
11	10	0.830037
12	11	0.851003
13	12	0.866975
14	13	0.872324
15	14	0.879461
16	15	0.885172



We find that we need 11 layers of the tree to get the 85% accuracy.



For the depth = 4:



It’s much simpler and also we still can get the 60% of accuracy.

IX. Discussion

By simplifying the model, we found that sometimes the number of parameters of the trained model can be reduced without very little penalty.

For the application of prediction, we was trying to achieve the goal that an user can enter any URL and then my application could give a prediction. However, we found that the result was not that correct, even a little ridiculous. I think it’s because of the randomness and arbitrary creation of the URL. For the given dataset, there might be many similar URL and they are in the past, there are tremendous new created URLs every day. It’s too naïve to predict the URL maliciousness purely by the lexical properties. And I’ll try to figure it out that how to make it achievable!

X. Conclusion

By this project, we know that we can use the Big data analysis method to distinguish which type an URL belongs and detect the suspicious URLs. Furthermore, in this project, we only need 4 features with the decision tree model of 11 layers to identify the type of URLs. However, it seem like that the research method is still not mature enough, I’ll try to investigate that in the future.