Configuration Manual

MSc Internship
Cyber Security

Dai Hoang Vu
Student ID: x17165423

School of Computing
National College of Ireland

Supervisor: Ben Fletcher

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Dai Hoang Vu		
Student ID:	X17165423		
Programme:	Cyber Security	**Year:**	2019
Module:	Internship		
Supervisor:	Ben Fletcher		
Submission Due Date:	12/12/2019		
Project Title:	Using Domain-Based on Machine Learning for Malware Detection		
Word Count:	4774	**Page Count:**	18

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.
I agree to an electronic copy of my thesis being made publicly available on NORMA the National College of Ireland's Institutional Repository for consultation.

Signature: Dai Hoang Vu

Date: 12/12/2019

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	x
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	x
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	x

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Using Domain-Based on Machine Learning for Malware Detection

Configuration Manual Submission

Dai Hoang Vu – X17165423

1. Install Tools

Step to Step for running the Machine Learning Algorithms to solves the problem about detecting Malicious Domain:

- Download and Install Anacoda[1] : https://www.anaconda.com/distribution/
- Find Cmd and activate tendor
- Download and Install PyCharm: https://www.jetbrains.com/pycharm/download/#section=windows
- Install Python 3.6
- In PyCharm, create new Project also go Setting, choose Project and click Existing interpreter, in here choose the folder which Anacoda was install and select python
- From the terminal of the Pycharm, you have to install some package like tensorflow, keras, skript,… and definitely install jupyterlab … all package easy install by the command pip install jupyter-lab[2][3].
- After installing every package you need, from the terminal again, type : jupyter-lab. The Pycharm will create the host and which send you to the Jupyter lab to start the code.
- Copy all data set (Good and bad domain) which you downloaded before to the folder called data inside the PycharmProject.
- From there just follow every step which I show in these pictures because in the code, I mentioned already which code use for what missions. Or just open the Pycharm and start the project with Jupyter-lab you will see whole process without typing code yourself
- Source Code Download Link : https://drive.google.com/file/d/1c-eXw1qs3XnyPSdjv76--7zhib7qjmXd/view

2. Start Coding

- Import Package : Import algorithms from the Scikit Learn library

Untitled.ipynb

Code | Python 3

```
Project : Using Domain-Based on Machine Learning for Malware Detection
NCI College
Dai Hoang Vu
Algorithms used :
- Random Forest
- Logistic Regression
- Naive Bayes

Data Gathering
Legit Domain : Alexa https://github.com/mozilla/cipherscan/tree/master/top1m
Malicious Domain :
360 Lab Domain (over 1m  domains)
https://data.netlab.360.com/feeds/dga/dga.txt
```

```
First Step : Set up
```

[1]:
```python
import numpy as np
import pandas as pd
import re
from publicsuffixlist import PublicSuffixList
import gc
import math
import collections
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_curve, auc, roc_auc_score
from keras.preprocessing import sequence
from keras.models import Sequential
```

- Import Good and Bad Domain also Combine it all in One

Untitled.ipynb

Code | Python 3

```python
RANDOM_SEED = 1

%matplotlib inline
```

```
Using TensorFlow backend.
```

```
Put Alexa 1 milions domain
```

[18]:
```python
good_domain = pd.read_csv('data/top-1m-domain.csv', header=None, names=['Domain'])
good_domain.head()
good_domain['DGA_Family'] = 'none'
good_domain['Type'] = 'Normal'
good_domain =good_domain[['DGA_Family','Domain','Type']]
good_domain.describe()
```

[18]:

	DGA_Family	Domain	Type
count	1000000	1000000	1000000
unique	1	1000000	1
top	none	zuk-media.com	Normal
freq	1000000	1	1000000

[19]:
```python
bad_domain = pd.read_table('data/360_dga.txt',names=['DGA_Family','Domain','Start_time','End_time'])
bad_domain = bad_domain.iloc[:, 0:2]
bad_domain['Type']='DGA'
bad_domain.to_csv('data/360_dga_domain.csv', index = False)
bad_domain.describe()
```

[19]:

	DGA_Family	Domain	Type
count	1169720	1169720	1169720
unique	42	1147770	1

Untitled.ipynb

```
AttributeError: 'float' object has no attribute 'split'
```

Make all Dommain in one file

```
[23]: bad_domain = pd.concat([bad_domain,bad1_domain],axis=0)
      bad_domain = bad_domain.drop_duplicates()
      all_domain = pd.concat([bad_domain,good_domain])
      all_domain.describe()
```

[23]:

	DGA_Family	Domain	Type
count	2183900	2183911	2183911
unique	64	2183847	2
top	none	laugqaoqixofl.ddns.net	DGA
freq	1000000	2	1183911

Save all data to csv file

```
[24]: all_domain_shuffle = all_domain.sample(frac = 1, random_state=RANDOM_SEED)
      all_domain_shuffle.to_csv('data/all_domain.csv', index = False)
```

```
[25]: all_domain_shuffle.head()
```

[25]:

	DGA_Family	Domain	Type
598985	none	ytsmovies.ga	Normal
669719	none	triger.com.pl	Normal
522313	none	fishintrepid.com	Normal
752250	banjori	nplramentalistfanchonut.com	DGA
331668	none	bikepricesinnepal.com	Normal

Untitled.ipynb

This section will show how we create the dataset about attributes for domain. It is include DGA name, lenght of domain(DNL), number of subdomain (NoS),subdomain length mean(SLM), have www Prefix(HwP), Has a Valid Top Level Domain (HVTLD), Contains Single-Character Subdomain (CSCS), Contains Top Level Domain as Subdomain (CTS),Underscore Ratio (UR), Contains IP address(CIPA), Contains digit (contains_digit), The ratiol of vowel(vowel_ration), The ratio of digit(digit_ratio), The ratio of repeated characters in a subdomain(RRC), The ratio of consecutive digits(RCD), The entropy of subdomain(Entropy)

Now create the dataset

```
[26]: domain_withFeatures = all_domain_shuffle.copy()
      domain_withFeatures.head()
```

[26]:

	DGA_Family	Domain	Type
598985	none	ytsmovies.ga	Normal
669719	none	triger.com.pl	Normal
522313	none	fishintrepid.com	Normal
752250	banjori	nplramentalistfanchonut.com	DGA
331668	none	bikepricesinnepal.com	Normal

```
[27]: # Load Valid Top Level Domains data
      import sys

      topLevelDomain = []
      with open('data/1.txt', 'r') as content:
          for line in content:
              topLevelDomain.append((line.strip('\n')))

      print(topLevelDomain)
```

```
['AAA', 'AARP', 'ABARTH', 'ABB', 'ABBOTT', 'ABBVIE', 'ABC', 'ABLE', 'ABOGADO', 'ABUDHABI', 'AC', 'ACADEMY', 'ACCENTURE', 'ACCOUNTANT', 'ACCOUNTANTS', 'ACO', 'ACTIV
E', 'ACTOR', 'AD', 'ADAC', 'ADS', 'ADULT', 'AE', 'AEG', 'AERO', 'AETNA', 'AF', 'AFAMILYCOMPANY', 'AFL', 'AFRICA', 'AG', 'AGAKHAN', 'AGENCY', 'AI', 'AIG', 'AIGO', 'AI
RBUS', 'AIRFORCE', 'AIRTEL', 'AKDN', 'AL', 'ALFAROMEO', 'ALIBABA', 'ALIPAY', 'ALLFINANZ', 'ALLSTATE', 'ALLY', 'ALSACE', 'ALSTOM', 'AM', 'AMERICANEXPRESS', 'AMERICANF
AMILY', 'AMEX', 'AMFAM', 'AMICA', 'AMSTERDAM', 'ANALYTICS', 'ANDROID', 'ANQUAN', 'ANZ', 'AO', 'AOL', 'APARTMENTS', 'APP', 'APPLE', 'AQ', 'AQUARELLE', 'AR', 'ARAB',
```

- Set up code for each attribute. This set-up code helps the computer to recognize values, so that the computer can learn and make future assessments. For example, the Type variable, we will divide the good Domain with the value 0 and the bad domain variable with the value 1.

Untitled.ipynb

Code Python 3

Set up code for each Attributes

[28]:
```python
psl = PublicSuffixList()

def ignoreVPS(domain):
    # Return the rest of domain after ignoring the Valid Public Suffixes:
    validPublicSuffix = '.' + psl.publicsuffix(domain)
    if len(validPublicSuffix) < len(domain):
         # If it has VPS
        subString = domain[0: domain.index(validPublicSuffix)]
    elif len(validPublicSuffix) == len(domain):
        return 0
    else:
        # If not
        subString = domain

    return subString

def typeTo_Binary(type):
  # Convert Type to Binary variable DGA = 1, Normal = 0
  if type == 'DGA':
    return 1
  else:
    return 0

def domain_length(domain):
  # Generate Domain Name Length (DNL)
  return len(domain)

def subdomains_number(domain):
  # Generate Number of Subdomains (NoS)
    subdomain = ignoreVPS(domain)
    return (subdomain.count('.') + 1)

def subdomain_length_mean(domain):
  # enerate Subdomain Length Mean (SLM)
```

Untitled.ipynb

Code Python 3

```python
def subdomain_length_mean(domain):
  # enerate Subdomain Length Mean (SLM)
    subdomain = ignoreVPS(domain)
    result = (len(subdomain) - subdomain.count('.')) / (subdomain.count('.') + 1)
    return result

def has_www_prefix(domain):
  # Generate Has www Prefix (HwP)
  if domain.split('.')[0] == 'www':
    return 1
  else:
    return 0

def has_hvltd(domain):
  # Generate Has a Valid Top Level Domain (HVTLD)
  if domain.split('.')[len(domain.split('.')) - 1].upper() in topLevelDomain:
    return 1
  else:
    return 0

def contains_single_character_subdomain(domain):
  # Generate Contains Single-Character Subdomain (CSCS)
    domain = ignoreVPS(domain)
    str_split = domain.split('.')
    minLength = len(str_split[0])
    for i in range(0, len(str_split) - 1):
        minLength = len(str_split[i]) if len(str_split[i]) < minLength else minLength
    if minLength == 1:
        return 1
    else:
        return 0

def contains_TLD_subdomain(domain):
  # Generate Contains TLD as Subdomain (CTS)
    subdomain = ignoreVPS(domain)
    str_split = subdomain.split('.')
    for i in range(0, len(str_split) - 1):
        if str_split[i].upper() in topLevelDomain:
```

Untitled.ipynb

Code Python 3

```python
    for i in range(0, len(str_split) - 1):
        if str_split[i].upper() in topLevelDomain:
            return 1
    return 0

def underscore_ratio(domain):
  # Generate Underscore Ratio (UR) on dataset
    subString = ignoreVPS(domain)
    result = subString.count('_') / (len(subString) - subString.count('.'))
    return result

def contains_IP_address(domain):
  # Generate Contains IP Address (CIPA) on datasetx
    splitSet = domain.split('.')
    for element in splitSet:
        if(re.match("\d+", element)) == None:
            return 0
    return 1

def contains_digit(domain):
    """
    Contains Digits
    """
    subdomain = ignoreVPS(domain)
    for item in subdomain:
        if item.isdigit():
            return 1
    return 0

def vowel_ratio(domain):
    """
    calculate Vowel Ratio
    """
    VOWELS = set('aeiou')
    v_counter = 0
    a_counter = 0
    ratio = 0
    subdomain = ignoreVPS(domain)
```

Untitled.ipynb

Code Python 3

```
    ratio = 0
    subdomain = ignoreVPS(domain)
    for item in subdomain:
        if item.isalpha():
            a_counter+=1
            if item in VOWELS:
                v_counter+=1
    if a_counter>1:
        ratio = v_counter/a_counter
    return ratio

def digit_ratio(domain):
    """
    calculate digit ratio
    """
    d_counter = 0
    counter = 0
    ratio = 0
    subdomain = ignoreVPS(domain)
    for item in subdomain:
        if item.isalpha() or item.isdigit():
            counter+=1
            if item.isdigit():
                d_counter+=1
    if counter>1:
        ratio = d_counter/counter
    return ratio

def prc_rrc(domain):
    """
    calculate the Ratio of Repeated Characters in a subdomain
    """
    subdomain = ignoreVPS(domain)
    subdomain = re.sub("[.]", "", subdomain)
    char_num=0
    repeated_char_num=0
    d = collections.defaultdict(int)
```

Untitled.ipynb

Code Python 3

```
    repeated_char_num=0
    d = collections.defaultdict(int)
    for c in list(subdomain):
        d[c] += 1
    for item in d:
        char_num +=1
        if d[item]>1:
            repeated_char_num +=1
    ratio = repeated_char_num/char_num
    return ratio

def prc_rcc(domain):
    """
    calculate the Ratio of Consecutive Consonants
    """
    VOWELS = set('aeiou')
    counter = 0
    cons_counter=0
    subdomain = ignoreVPS(domain)
    for item in subdomain:
        i = 0
        if item.isalpha() and item not in VOWELS:
            counter+=1
        else:
            if counter>1:
                cons_counter+=counter
            counter=0
        i+=1
    if i==len(subdomain) and counter>1:
        cons_counter+=counter
    ratio = cons_counter/len(subdomain)
    return ratio

def prc_rcd(domain):
    """
    calculate the ratio of consecutive digits
    """
    counter = 0
```

Untitled.ipynb

Code Python 3

```
    ratio = cons_counter/len(subdomain)
    return ratio

def prc_rcd(domain):
    """
    calculate the ratio of consecutive digits
    """
    counter = 0
    digit_counter=0
    subdomain = ignoreVPS(domain)
    for item in subdomain:
        i = 0
        if item.isdigit():
            counter+=1
        else:
            if counter>1:
                digit_counter+=counter
            counter=0
        i+=1
    if i==len(subdomain) and counter>1:
        digit_counter+=counter
    ratio = digit_counter/len(subdomain)
    return ratio

def prc_entropy(domain):
    """
    calculate the entropy of subdomain
    :param domain_str: subdomain
    :return: the value of entropy
    """
    subdomain = ignoreVPS(domain)
    # get probability of chars in string
    prob = [float(subdomain.count(c)) / len(subdomain) for c in dict.fromkeys(list(subdomain))]

    # calculate the entropy
    entropy = - sum([p * math.log(p) / math.log(2.0) for p in prob])
    return entropy
```

- Create Feature for Domain Attributes: Remove some attribute not important for ML algorithms

```python
[29]: #Create each feature
def extract_features():
    domain_withFeatures['DNL'] = domain_withFeatures['Domain'].apply(lambda x: domain_length(x))
    domain_withFeatures['NoS'] = domain_withFeatures['Domain'].apply(lambda x: subdomains_number(x))
    domain_withFeatures['SLM'] = domain_withFeatures['Domain'].apply(lambda x: subdomain_length_mean(x))
    domain_withFeatures['HwP'] = domain_withFeatures['Domain'].apply(lambda x: has_www_prefix(x))
    domain_withFeatures['HVTLD'] = domain_withFeatures['Domain'].apply(lambda x: has_hvltd(x))
    domain_withFeatures['CSCS'] = domain_withFeatures['Domain'].apply(lambda x: contains_single_character_subdomain(x))
    domain_withFeatures['CTS'] = domain_withFeatures['Domain'].apply(lambda x: contains_TLD_subdomain(x))
    domain_withFeatures['UR'] = domain_withFeatures['Domain'].apply(lambda x: underscore_ratio(x))
    domain_withFeatures['CIPA'] = domain_withFeatures['Domain'].apply(lambda x: contains_IP_address(x))
    domain_withFeatures['contains_digit']= domain_withFeatures['Domain'].apply(lambda x:contains_digit(x))
    domain_withFeatures['vowel_ratio']= domain_withFeatures['Domain'].apply(lambda x:vowel_ratio(x))
    domain_withFeatures['digit_ratio']= domain_withFeatures['Domain'].apply(lambda x:digit_ratio(x))
    domain_withFeatures['RRC']= domain_withFeatures['Domain'].apply(lambda x:prc_rrc(x))
    domain_withFeatures['RCC']= domain_withFeatures['Domain'].apply(lambda x:prc_rcc(x))
    domain_withFeatures['RCD']= domain_withFeatures['Domain'].apply(lambda x:prc_rcd(x))
    domain_withFeatures['Entropy']= domain_withFeatures['Domain'].apply(lambda x:prc_entropy(x))
```

```python
[30]: # Generate features
extract_features()
```

```python
[31]: # Change Type virable from DGA and Normal to 1 and 0
domain_withFeatures['Type'] = domain_withFeatures['Type'].apply(lambda x: typeTo_Binary(x))
```

Now head to process the data

```python
[32]: domain_withFeatures.head()
```

[32]:

	DGA_Family	Domain	Type	DNL	NoS	SLM	HwP	HVTLD	CSCS	CTS	UR	CIPA	contains_digit	vowel_ratio	digit_ratio	RRC	RCC	RCD	Entropy
598985	none	ytsmovies.ga	0	12	1	9.0	0	1	0	0	0.0	0	0	0.333333	0.0	0.125000	0.444444	0.0	2.947703
669719	none	triger.com.pl	0	13	1	6.0	0	1	0	0	0.0	0	0	0.333333	0.0	0.200000	0.333333	0.0	2.251629
522313	none	fishintrepid.com	0	16	1	12.0	0	1	0	0	0.0	0	0	0.333333	0.0	0.100000	0.416667	0.0	3.188722
752250	banjori	nplramentalistfanchonut.com	1	27	1	23.0	0	1	0	0	0.0	0	0	0.304348	0.0	0.266667	0.521739	0.0	3.675311

```python
[33]: domain_withFeatures.describe()
```

[33]:

	Type	DNL	NoS	SLM	HwP	HVTLD	CSCS	CTS	UR	CIPA	contains_digit	vowel_ratio	digit_ratio	
count	2.183911e+06	2.183911e+06	2.183911e+06	2.183911e+06	2.183911e+06	2.183911e+06	2.183911e+06	2.183911e+06	2.183911e+06	2183911.0	2.183911e+06	2.183911e+06	2.183911e+06	2.1
mean	5.421059e-01	1.796130e+01	1.014205e+00	1.383345e+01	5.403151e-05	9.998649e-01	6.776833e-05	1.057735e-04	5.711692e-07	0.0	1.305612e-01	3.060640e-01	3.405578e-02	2.
std	4.982241e-01	5.526419e+00	1.186975e-01	5.451104e+00	7.350416e-03	1.162156e-02	8.231875e-03	1.028408e-02	2.088700e-04	0.0	3.369199e-01	1.322936e-01	1.002824e-01	1.
min	0.000000e+00	2.000000e+00	1.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.0	0.000000e+00	0.000000e+00	0.000000e+00	0.(
25%	0.000000e+00	1.400000e+01	1.000000e+00	1.000000e+01	0.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.0	0.000000e+00	2.142857e-01	0.000000e+00	1.
50%	1.000000e+00	1.900000e+01	1.000000e+00	1.400000e+01	0.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.0	0.000000e+00	3.181818e-01	0.000000e+00	2.
75%	1.000000e+00	2.100000e+01	1.000000e+00	1.800000e+01	0.000000e+00	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.0	0.000000e+00	4.000000e-01	0.000000e+00	3.
max	1.000000e+00	7.300000e+01	4.000000e+00	6.300000e+01	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.538462e-01	0.0	1.000000e+00	1.000000e+00	1.000000e+00	1.(

```python
[34]: # Save the data
domain_withFeatures.to_csv('data/domain_withFeatures.csv', index=False)
```

```python
[35]: # Load the data again
domain_withFeatures = pd.read_csv('data/domain_withFeatures.csv')
```

```python
[36]: domain_withFeatures.head()
```

[36]:

	DGA_Family	Domain	Type	DNL	NoS	SLM	HwP	HVTLD	CSCS	CTS	UR	CIPA	contains_digit	vowel_ratio	digit_ratio	RRC	RCC	RCD	Entropy
0	none	ytsmovies.ga	0	12	1	9.0	0	1	0	0	0.0	0	0	0.333333	0.0	0.125000	0.444444	0.0	2.947703
1	none	triger.com.pl	0	13	1	6.0	0	1	0	0	0.0	0	0	0.333333	0.0	0.200000	0.333333	0.0	2.251629
2	none	fishintrepid.com	0	16	1	12.0	0	1	0	0	0.0	0	0	0.333333	0.0	0.100000	0.416667	0.0	3.188722
3	banjori	nplramentalistfanchonut.com	1	27	1	23.0	0	1	0	0	0.0	0	0	0.304348	0.0	0.266667	0.521739	0.0	3.675311
4	none	bikepricesinnepal.com	0	21	1	17.0	0	1	0	0	0.0	0	0	0.411765	0.0	0.363636	0.235294	0.0	3.292770

- Create the table: the values with blue color on the table show little correlation between the two variables, we will remove these variables to reduce redundancy for the algorithms we apply next.

Untitled.ipynb

4	none	bikepricesinnepal.com	0	21	1	17.0	0	1	0	0	0.0	0	0	0.411765	0.0	0.363636	0.235294	0.0	3.292770

Create a correlation map to show correlation between 2 attributes

[37]:
```
corrmat = domain_withFeatures.corr()

plt.figure(figsize=(15,15))
sns.heatmap(corrmat, annot=True, cmap= "RdBu_r")

k = 10 #number of variables for heatmap
cols = corrmat.nlargest(k, 'Type')['Type'].index
cm = np.corrcoef(domain_withFeatures[cols].values.T)
f, ax = plt.subplots(figsize=(16, 16))
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cmap = "RdBu_r", cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, yticklabels=cols.values, xticklabels=cols.values)
plt.show()
```

Untitled.ipynb

As we seen in the Map HwP, HVTLD, UR, CSCS has low correlation value so we will drop this value,also DGA_family and Domain we don't need for next step so we wil drop it as well

[41]:
```
drop_column = {'DGA_Family', 'Domain', 'HwP', 'HVTLD', 'UR', 'CSCS'}
```

[42]:
```
# Drop  columns
domain_withFeatures_fixed = domain_withFeatures.drop(drop_column, axis = 1)
domain_withFeatures_fixed.head()
```

[42]:

	Type	DNL	NoS	SLM	CTS	CIPA	contains_digit	vowel_ratio	digit_ratio	RRC	RCC	RCD	Entropy
0	0	12	1	9.0	0	0	0	0.333333	0.0	0.125000	0.444444	0.0	2.947703
1	0	13	1	6.0	0	0	0	0.333333	0.0	0.200000	0.333333	0.0	2.251629
2	0	16	1	12.0	0	0	0	0.333333	0.0	0.100000	0.416667	0.0	3.188722
3	1	27	1	23.0	0	0	0	0.304348	0.0	0.266667	0.521739	0.0	3.675311
4	0	21	1	17.0	0	0	0	0.411765	0.0	0.363636	0.235294	0.0	3.292770

[43]:
```
domain_withFeatures_fixed.describe()
```

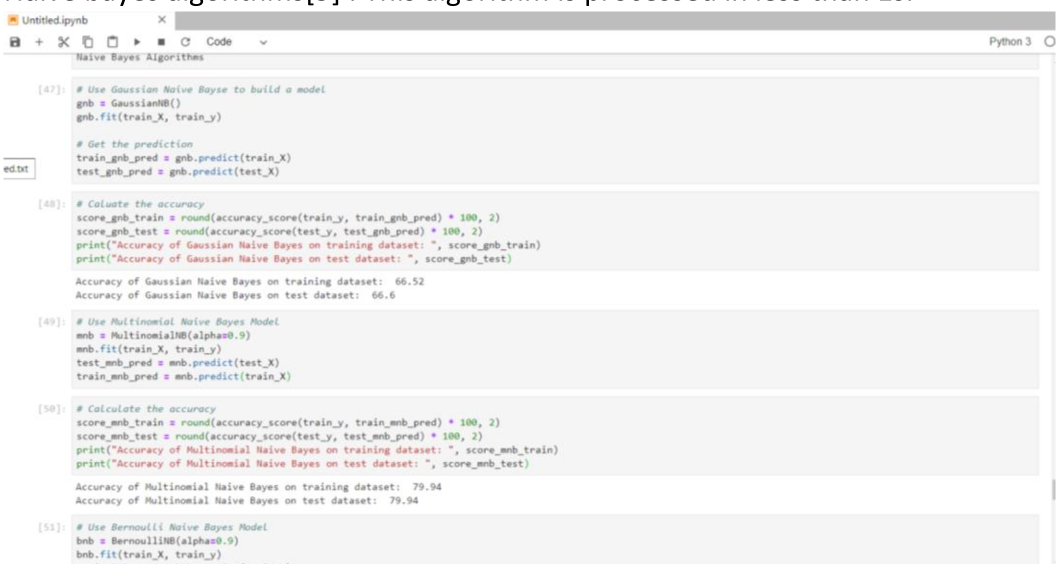
[43]:

	Type	DNL	NoS	SLM	CTS	CIPA	contains_digit	vowel_ratio	digit_ratio	RRC	RCC	RCD	Entropy
count	2.183911e+06	2.183911e+06	2.183911e+06	2.183911e+06	2.183911e+06	2183911.0	2.183911e+06	2.183911e+06	2.183911e+06	2.183911e+06	2.183911e+06	2.183911e+06	2.183911e+06
mean	5.421059e-01	1.796130e+01	1.014205e+00	1.383345e+01	1.057735e-04	0.0	1.305612e-01	3.060640e-01	3.405578e-02	2.667058e-01	3.764959e-01	1.285510e-02	3.172528e+00
std	4.982241e-01	5.526419e+00	1.186975e-01	5.451104e+00	1.028408e-02	0.0	3.369199e-01	1.322936e-01	1.002824e-01	1.597517e-01	2.084597e-01	5.491473e-02	5.459297e-01
min	0.000000e+00	2.000000e+00	1.000000e+00	1.000000e+00	0.000000e+00	0.0	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	-0.000000e+00
25%	0.000000e+00	1.400000e+01	1.000000e+00	1.000000e+01	0.000000e+00	0.0	0.000000e+00	2.142857e-01	0.000000e+00	1.538462e-01	2.500000e-01	0.000000e+00	2.855389e+00
50%	1.000000e+00	1.900000e+01	1.000000e+00	1.400000e+01	0.000000e+00	0.0	0.000000e+00	3.181818e-01	0.000000e+00	2.500000e-01	3.913043e-01	0.000000e+00	3.272906e+00
75%	1.000000e+00	2.100000e+01	1.000000e+00	1.800000e+01	0.000000e+00	0.0	0.000000e+00	4.000000e-01	0.000000e+00	3.636364e-01	5.000000e-01	0.000000e+00	3.577820e+00
max	1.000000e+00	7.300000e+01	4.000000e+00	6.300000e+01	1.000000e+00	0.0	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	9.583333e-01	9.000000e-01	4.954196e+00

- Prepare variable for Training Dataset and Testing Dataset[4]

```
Untitled.ipynb
Code                                                                                     Python 3
     75%  1.000000e+00  2.100000e+01  1.000000e+00  1.800000e+01  0.000000e+00  0.0  0.000000e+00  4.000000e-01  0.000000e+00  3.636364e-01  5.000000e-01  0.000000e+00  3.577820e+00
     max  1.000000e+00  7.300000e+01  4.000000e+00  6.300000e+01  1.000000e+00  0.0  1.000000e+00  1.000000e+00  1.000000e+00  1.000000e+00  9.583333e-01  9.000000e-01  4.954196e+00

[44]: domain_withFeatures_fixed.isnull().sum()

[44]: Type              0
      DNL               0
      NoS               0
      SLM               0
      CTS               0
      CIPA              0
      contains_digit    0
      vowel_ratio       0
      digit_ratio       0
      RRC               0
      RCC               0
      RCD               0
      Entropy           0
      dtype: int64

      Prepair for training and testing dataset

[45]: # Show independent variables and dependent variables
      attributes = domain_withFeatures_fixed.drop('Type', axis=1)
      observed = domain_withFeatures_fixed['Type']
      attributes.shape, observed.shape

[45]: ((2183911, 12), (2183911,))

[46]: # Split the dataset into training dataset and testing dataset
      train_X, test_X, train_y, test_y = train_test_split(attributes, observed, test_size = 0.25, random_state = RANDOM_SEED)
      train_X.shape, test_X.shape, train_y.shape, test_y.shape

[46]: ((1637933, 12), (545978, 12), (1637933,), (545978,))
```

- Naïve bayes algorithms[5] : This algorithm is processed in less than 1s.



```
Untitled.ipynb
Code                                                                                     Python 3
      Naive Bayes Algorithms

[47]: # Use Gaussian Naive Bayse to build a model
      gnb = GaussianNB()
      gnb.fit(train_X, train_y)

      # Get the prediction
      train_gnb_pred = gnb.predict(train_X)
      test_gnb_pred = gnb.predict(test_X)

[48]: # Caluate the accuracy
      score_gnb_train = round(accuracy_score(train_y, train_gnb_pred) * 100, 2)
      score_gnb_test = round(accuracy_score(test_y, test_gnb_pred) * 100, 2)
      print("Accuracy of Gaussian Naive Bayes on training dataset: ", score_gnb_train)
      print("Accuracy of Gaussian Naive Bayes on test dataset: ", score_gnb_test)

      Accuracy of Gaussian Naive Bayes on training dataset:  66.52
      Accuracy of Gaussian Naive Bayes on test dataset:  66.6

[49]: # Use Multinomial Naive Bayes Model
      mnb = MultinomialNB(alpha=0.9)
      mnb.fit(train_X, train_y)
      test_mnb_pred = mnb.predict(test_X)
      train_mnb_pred = mnb.predict(train_X)

[50]: # Calculate the accuracy
      score_mnb_train = round(accuracy_score(train_y, train_mnb_pred) * 100, 2)
      score_mnb_test = round(accuracy_score(test_y, test_mnb_pred) * 100, 2)
      print("Accuracy of Multinomial Naive Bayes on training dataset: ", score_mnb_train)
      print("Accuracy of Multinomial Naive Bayes on test dataset: ", score_mnb_test)

      Accuracy of Multinomial Naive Bayes on training dataset:  79.94
      Accuracy of Multinomial Naive Bayes on test dataset:  79.94

[51]: # Use Bernoulli Naive Bayes Model
      bnb = BernoulliNB(alpha=0.9)
      bnb.fit(train_X, train_y)
      train_bnb_pred = bnb.predict(train_X)
```

- Logistic Regression Algorithms[6]: This algorithm is processed in 30 seconds.

```
Untitled.ipynb
Code                                                    Python 3
[51]: # Use Bernoulli Naive Bayes Model
      bnb = BernoulliNB(alpha=0.9)
      bnb.fit(train_X, train_y)
      train_bnb_pred = bnb.predict(train_X)
      test_bnb_pred = bnb.predict(test_X)

[52]: # Calculate the accuracy
      score_bnb_train = round(accuracy_score(train_y, train_bnb_pred) * 100, 2)
      score_bnb_test = round(accuracy_score(test_y, test_bnb_pred) * 100, 2)
      print("Accuracy of Bernoulli Naive Bayes on training dataset: ", score_bnb_train)
      print("Accuracy of Bernoulli Naive Bayes on test dataset: ", score_bnb_test)

      Accuracy of Bernoulli Naive Bayes on training dataset:  62.73
      Accuracy of Bernoulli Naive Bayes on test dataset:  62.71

      Logistic Regression

[53]: # Use Logisitic Regression to build the model
      lg = LogisticRegression(random_state=RANDOM_SEED)
      lg.fit(train_X, train_y)

      train_lg_pred = lg.predict(train_X)
      test_lg_pred = lg.predict(test_X)

      C:\Users\hoang\Anaconda3\envs\tensor\lib\site-packages\sklearn\linear_model\_logistic.py:939: ConvergenceWarning: lbfgs failed to converge (status=1):
      STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

      Increase the number of iterations (max_iter) or scale the data as shown in:
          https://scikit-learn.org/stable/modules/preprocessing.html.
      Please also refer to the documentation for alternative solver options:
          https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
        extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)

[54]: # Calculate the accuracy
      score_lg_train = round(accuracy_score(train_y, train_lg_pred) * 100, 2)
      score_lg_test = round(accuracy_score(test_y, test_lg_pred) * 100, 2)
      print("Accuracy of Logistic Regression on training dataset: ", score_lg_train)
      print("Accuracy of Logistic Regression on test dataset: ", score_lg_test)
```

- Random Forest Algorithms[7]: This algorithm is processed in 5 minutes and 15 seconds.

```
Untitled.ipynb
Code                                                    Python 3
      print("Accuracy of Logistic Regression on training dataset: ", score_lg_train)
      print("Accuracy of Logistic Regression on test dataset: ", score_lg_test)

      Accuracy of Logistic Regression on training dataset:  84.92
      Accuracy of Logistic Regression on test dataset:  84.91

      Random Forest

[55]: # Set up the Random Forest Model
      rf = RandomForestClassifier(random_state= RANDOM_SEED)

      # Train the model with training data
      rf.fit(train_X, train_y)

[55]: RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                             criterion='gini', max_depth=None, max_features='auto',
                             max_leaf_nodes=None, max_samples=None,
                             min_impurity_decrease=0.0, min_impurity_split=None,
                             min_samples_leaf=1, min_samples_split=2,
                             min_weight_fraction_leaf=0.0, n_estimators=100,
                             n_jobs=None, oob_score=False, random_state=1, verbose=0,
                             warm_start=False)

[56]: # Get the prediction
      train_rf_pred = rf.predict(train_X)
      test_rf_pred = rf.predict(test_X)

[57]: # Calculate the accuracy
      score_rf_train = round(accuracy_score(train_y, train_rf_pred) * 100, 2)
      score_rf_test = round(accuracy_score(test_y, test_rf_pred) * 100, 2)
      print("Accuracy of Random Forest Model on training dataset: ", score_rf_train)
      print("Accuracy of Random Forest Model on test dataset: ", score_rf_test)

      Accuracy of Random Forest Model on training dataset:  93.81
      Accuracy of Random Forest Model on test dataset:  93.44

[ ]:
```

Thanks for watching my tutorial

3. References :

[1] Docs.anaconda.com. (2020). *User guide — Anaconda documentation*. [online] Available at: https://docs.anaconda.com/anaconda/user-guide/ [Accessed 1 Dec. 2019].
[2]"JupyterLab Documentation — JupyterLab 1.2.6 documentation", *Jupyterlab.readthedocs.io*, 2020. [Online]. Available: https://jupyterlab.readthedocs.io/en/stable/. [Accessed: 02- Dec- 2019].
[3]"scikit-learn: machine learning in Python — scikit-learn 0.22.1 documentation", *Scikit-learn.org*, 2020. [Online]. Available: https://scikit-learn.org/stable/. [Accessed: 01- Dec- 2019].
[4]S. Srinidhi, "How to split your dataset to train and test datasets using SciKit Learn", *Medium*, 2020. [Online]. Available: https://medium.com/@contactsunny/how-to-split-your-dataset-to-train-and-test-datasets-using-scikit-learn-e7cf6eb5e0d. [Accessed: 01- Dec- 2020].

[5]A. Navlani, "Naive Bayes Classification using Scikit-learn", *DataCamp Community*, 2020. [Online]. Available: https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn. [Accessed: 02- Dec- 2019].
[6]S. Li, "Building A Logistic Regression in Python, Step by Step", *Medium*, 2020. [Online]. Available: https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8. [Accessed: 01- Dec- 2019].
[7]W. Koehrsen, "An Implementation and Explanation of the Random Forest in Python", *Medium*, 2020. [Online]. Available: https://towardsdatascience.com/an-implementation-and-explanation-of-the-random-forest-in-python-77bf308a9b76. [Accessed: 01- Dec- 2019].