Honours Project Final Project

Name: Fengyuan Zhang

Supervisor: Dr Miguel Garzon

Phishing website detection using machine learning

Abstract

With the development network and the population of computers and smart devices, more and more people prefer to choose online activities such as shopping, booking, ordering, applying. However, dangerous always hide in common convenient activities. Cybersecurity is one of the dangers which is related to user training, propaganda and technique protection. However, the effects or results always depend on the weakest board. The propaganda and technique protection is supported by relative companies that are full of professional practitioners so the users are the easiest to be broken through. In this case, our project is designed to provide a basic method to protect users' security. Using machine learning to help users to find which web might be phishing web to prevent their losses.

1. Introduction

Our project is designed for users who wary about scam and information or economy losses caused by phishing websites but without enough knowledge and related experience. We collect phishing websites and normal websites to train the AI machine. Our phishing websites dataset is from Phishingtank. Our normal website is from Moz which is a network statistics service company. Besides these, we also use Urlscan and Shodan to scan the websites which we have collected to get their information for our AI training.

1.1. Difficulties and challenges

As the project progresses, more and more difficulties surfaced. First, scanning websites can only provide statistics numbers. The details and scripts behind those webs are not available through scanning webs. To solve this problem, I choose to save the web as a text file in the local disk and use python to do relative jobs such as search keyword. Second, because the activities of phishing webs are illegal, their life cycle is very short. In another word, we can still get their statistic numbers from the scanning web record but we cannot access their web anymore. For those websites, we cannot use them anymore to train our AI machine. Third, since we have over 9000 phishing websites and 1000 normal websites and for each of those websites we all need to scan in two sites and process locally, we have to be patient. For every website, it needs different time

to process. For some web, my python script only needs 4 seconds to get the result, however for some it needs more than 5 minutes. By now we do not have a better method to alternative it so we can only wait until it ends.

1.2. Tools used

In this project, we only use python3 with its different packets and different api to develop. To achieve the goal of automat scanning, We use python and api from Urlscan and Shodan to upload and save result. To train AI model, we use scikit-learn flow to get what we want.

1.3. Usage

To achieve what we want, we separate the code as two parts. The first one is property extract. Its use age as follow:

Python generate_features.py -f listofWebsites.

After we type this in terminal, it will generate a csv file of properties of list of web. After it we, we use the file to train AI model.

Type python Phshing_or_not web in terminal to determine if the web a phishing web.

2. AI training

In this project, we choose to use feedforward neural network (ANN), artificial neutron, as our AI model.

2.1. Library choose

To make sure that we can do the training, we choose to use scikit-learn as our library. First, it is developed by Google which means it is supported by professional team. Hence its performance and stability can be guaranteed [1]. Second, scikit-learn is easier to debug since it allow us to execute subparts of a graph [2]. Last but not least, it allows pipeline [2]. As mentioned before, we have over 9000 phishing websites and 1000 normal websites, with the help of pipeline we can shrink the loading time.

2.3. Professional terms

2.3.1. Artificial neural networks (ANN)

Artificial neural networks are software which mimic neuronal structure of our brains. Our brains are full of neurones which work as switches. When the input signal is send to one neurone, it would be automatically processed and sent to other neurones. At the beginning, the ANN is designed to perform missions as our brain. Now, it has been used on some specific tasks such as computer vision, speech recognition, filtering and others.

2.3.1.1Components of ANN

2.3.1.1.1 neurones

Artificial neurones is the basic unit of ANN. The artificial neurones receives one or more inputs and sums them to produce an output. This out put might go to

other layers may be the same. Usually, every input plays different roles in the process hence their weights are different [4].

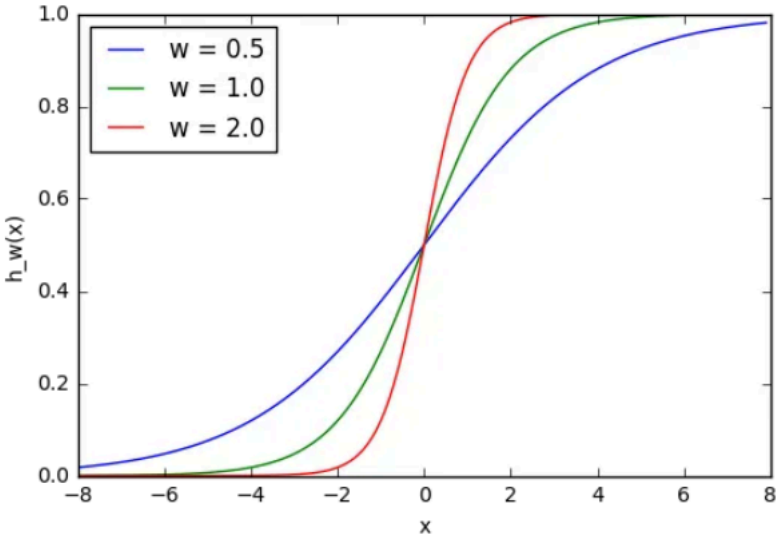
2.3.1.1.2 Connections and weights

The ANN consists of connections, however as mentioned above each connection has difference importance. In this case we need a scalar to determine the importance which is named as weight. For each neurone, it can has multiple inputs and outputs [3].

2.3.1.1.3 Bias

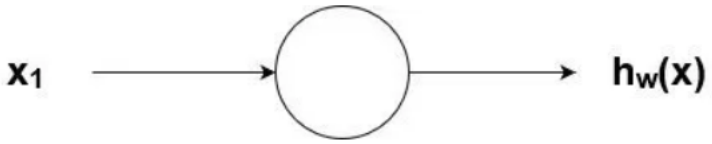
Consider the following python code and picture.

```
w1 = 0.5
w2 = 1.0
w3 = 2.0
l1 = 'w = 0.5'
l2 = 'w = 1.0'
l3 = 'w = 2.0'
for w, l in [(w1, l1), (w2, l2), (w3, l3)]:
    f = 1 / (1 + np.exp(-x*w))
    plt.plot(x, f, label=l)
plt.xlabel('x')
plt.ylabel('h_w(x)')
plt.legend(loc=2)
plt.show()
```

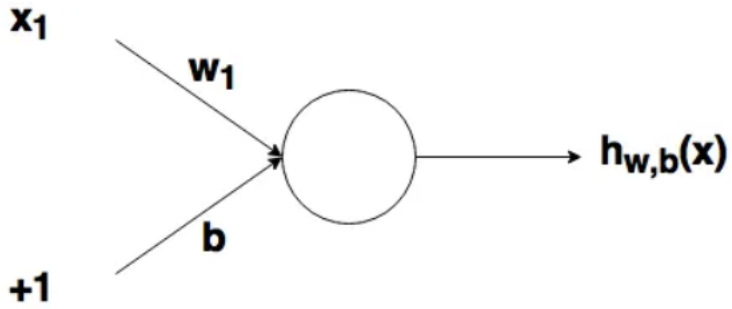


The above 2 pictures show that the changes of h_w(x) with different x. However if we only want to get the value of x the above figures cannot proved any help. In this case, we need a bias to help us.

The figure of a neurone without bias as follow:



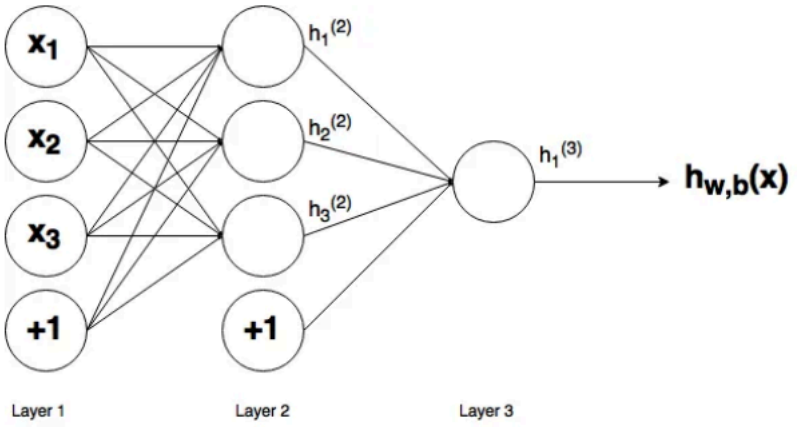
In comparison, the figure of a neurone with bias as follow:



[5]

2.3.2 ANN structure

Most common structure consists of input layer, hidden layer and output layer as shown:



The function of input layer(L1) and output layer(L3) as their name states, take input from outside and give result to outside. L2 is Called hidden layer, which is not belong to L1 or L2. Compare L2 to L1 and L3, there can exist multiple L2 for more complex training or calculation. Because of the difficulty of our project, we only use one L2 for our hidden layer.

3. Process of project

In this section, we separate the code into two parts to explain our project. The first part is the extraction of information of website. The second one is AI part. The benefit of this is that to make sure the information part and AI part would not influence each other. For example, if I want to fix the extraction code, I do not have to make the whole project restart again.

3.1. Extract information of web

3.1.1. Process order of the script

In the extraction phases, the script follow the given order. First, read a web from web list. Second upload the to urlscan web and get a uuid. Third save the uuid in the uuid.txt file. Then loop from the first step until it is end. Fourth, open the uuid.txt which is full of uuid of the result from urlscan, and upload the uuid to urlscan again to get the result of scanning, and extract information, and save in the csv format file. Fifth, using the url we get from urlscan, upload to shodan, to get more details about the web and save in the csv file as well. Last but not the least, we use the python installed command to delete the uuid.txt file which full of uuid of webs.

3.1.2. Import of packets

To make sure the script would extract information what we want, we import two series of packets. One is original installed in python, the other one is installed by pip command. Most of the third part packets are api from webs and tools such as urlscan, shodan and whois.

3.1.3 Parsing of the script

In this part we are going to parsing the code of the extraction part.

```
web_list_path = ''
current_path = os.path.abspath('.')
try:
    opts, args = getopt.getopt(argv, 'f:')
except getopt.GetoptError:
    print ('Error')
    sys.exit(2)
for opt, arg in opts:
    if opt in ('-f'):
        web_list_path = arg
print("web_list_path = " + web_list_path)
print("current_path = " + current_path)
```

The function of the above code is to make the script can be used in terminal as:

Python generate_features.py -f web list.txt.

As it's shown, we need to have a current path to create the uuid.txt text file in the following septs and delete it.

The function of above code is to read the web from text file and upload to the urlscan to get the uuid and save it to the uuid.txt file.

This figure is the csv file head, which are the properties what we want to extract from a give web. Besides this, the last property is 'Phishing' which is stetted as 1 (or True). The reason is that, we are doing the phishing website scanning, phishing property is known so we set it as 1 (or True). In the normal web scanning turn, we set it as 0 (or False).

```
        request_info = content.get("data").get("requests")
        verdict_info = content.get("verdicts")
        list_info = content.get("lists")                                    t_percentage',
        stats_info = content.get("stats")
        page_info = content.get("page")


    data = '{"url": "%s", "public": "on"}' % target_url

    response = requests.post('https://urlscan.io/api/v1/scan/', headers=headers, data=data.encode('utf-8'))

    if (response.status_code != 200):
        print("Server didn't return an 'OK' response.  Content was: {!r}".format(response.content))
        continue
    else:
        print(response.json())
        ## end POST request
        print()
        print("Working on file number: " + str(processing_number))
        if response.json().get("uuid") is None:
            continue
        uuid = response.json().get("uuid")
        print("print uuid:")
        print(uuid)
        uuidList.write(uuid + "\n")
        processing_number = processing_number + 1

        time.sleep(2)
uuidList.close()
```

This section of code is to parse the structure of the result from urlscan api.

```
        ### data for summary
        print('get data for summary')
        page_domain = page_info.get("domain")
        page_ip = page_info.get("ip")
        if (page_ip is None):
            continue
        page_country = page_info.get("country")
        page_server = page_info.get("server")
        ptr= page_info.get('ptr')
        asn= page_info.get('asn')
        asnname = page_info.get('asnname')
        ads_blocked = stats_info.get("adBlocked")
        securePercentage = stats_info.get("securePercentage")
        secureRequests= stats_info.get("secureRequests")
        ipv6_percentage = stats_info.get("IPv6Percentage")
        totalLinks= stats_info.get("totalLinks")
        score=verdict_info.get("overall").get("score")

        if verdict_info.get("overall").get("malicious") == True:
            is_malicious = 1
        else:
            is_malicious = 0

        urls = page_info.get("url")
        certificate= list_info.get("certificates")
        certificates=None
        if(certificate is None):
            certificates= 0
        else:
            certificates=1

        domurl=  content.get("task").get("domURL")
        domContent =requests.get(domurl).text
        print("Working on file number: " + str(fileNum))
```

The above code in figure is to get properties which we want after we parsing the structure of urlscan api result.

```
    #print(domContent)
    try:
        dom = hashlib.md5(domContent.encode()).hexdigest()
        print("Hash: " + dom)
        print(' ')
        page_protocol = protocol_info[0].get("protocol")
    except:
        page_protocol = 0

    #get favicon hash
    try:
        response1 = requests.get("https://www.google.com/s2/favicons?domain="+page_domain)
        favicon = response1.content
        favicon_hash_value = hashlib.md5(favicon).hexdigest()
    except:
        favicon_hash_value = 0
    print('favicon hash done')


    #get age of domain and validatity period
    try:
        domain = whois.query(page_domain)
        domain_age = (datetime.now() - domain.creation_date).total_seconds
        validatity_period = domain.__dict__.get("expiration_date")
        date_o = time.strptime(validatity_period, '%Y-%m-%d %H:%M:%S')
        validatity_period_time_stamp = int(time.mktime(date_o))
    except:
        domain_age = 0
        validatity_period_time_stamp = 0
    print('age calculated')
```

```
        #check dns record(MX, NS, SOA,CNAME)
        try:
            MX = dns.resolver.query(page_domain, "MX")
        except:
            MX = 0
        try:
            NS = dns.resolver.query(page_domain, "NS")
        except:
            NS = 0
        try:
            SOA = dns.resolver.query(page_domain, "SOA")
        except:
            SOA = 0
        print('record part done')
```

As mentioned in pervious, urlscan cannot provide all information as we want. In

this case, we need to use extra api to help us. The above code is used for this situation. The dom and favicon hash value are not provided in the scanning web, so we can only extract the original value and hash them by ourselves. The same problem for domain period and dns record. We can only solve them using extra api from other tools.

```
#get iframe and a tag numbers
try:
    req = urllib.request.Request(urls)
    webpage = urllib.request.urlopen(req)
    html = webpage.read()
    soup = BeautifulSoup(html, 'html.parser')
    iframe_number = (len(soup.find_all('iframe')))
    a_tag_number = (len(soup.find_all('a')))

    meta_link_number = (len(soup.find_all('meta')['href']))
    if (totalLinks != 0):
        meta_link_percentage = meta_link_number / totalLinks
    else:
        meta_link_percentage = 0
except:
    iframe_number = 0
    a_tag_number = 0
    a_tag_url_number_percentage = 0
    meta_link_percentage = 0
print('meta tage done')
```

```
#get mailto numbers
wb = xlwt.Workbook()
ws = wb.add_sheet('Emails')
ws.write(0,0,'Emails')
emailList = []
mailtos_number = 0
try:
    getH = requests.get(urls)
    h = getH.content
    soup = BeautifulSoup(h,'html.parser')
    mailtos = soup.select('[href^=mailto]')
    for i in mailtos:
        href = i['href']
        try:
            str1, str2 = href.split(':')
        except ValueError:
            break
        emailList.append(str2)
        mailtos_number = len(emailList)
except:
    mailtos_number = 0
print('mailto done')
```

The above two codes are about iframe and tag in web and mailto function hidden in html. Clearly, they also need help from other tools, one is urllib another is xlwt.

```
try:
    if page_domain in urls:
        domain_in_url = 1
    else:
        domain_in_url = 0
except:
    domain_in_url = 0
```

This above two code is the solution of the limitation of scanning web. As we mentioned before, some information cannot access from web, so we download the html as text file and use python to process (The second one). This is a

```
#check if right click disable, hover change status bar
right_disable = 0
hover_change_status_bar = 0
contain_popup_window = 0
try:
    r = requests.get(urls)
    txt = r.text
    with open (current_path + "/temple.txt","w") as f:
        f.write(txt)
        f.close()       #save page source code
    with open (current_path + "/temple.txt") as data:
        if "event.button==2" in data.read():
            right_disable = 1
        if "onMouseover=" + '"window.status=' in data.read():
            hover_change_status_bar = 1
            data.close()    #search from the source code
        if 'alert(' | 'toggle(' in data.read():
            contain_popup_window = 1
except:
    right_disable = 0
    hover_change_status_bar = 0
    contain_popup_window = 0
print('right click and hover part done')
```

controversial solution, because we save it as a string in text file when we search key word which means python might mistake keywords in paragraph as function keyword.

```
    #check if indexed by google
    indexed_by_google = 0
    try:
        request = requests.get(urls)
        if request.status_code == 200:
            indexed_by_google = 1
        else:
            indexed_by_google = 0
    except:
        indexed_by_google = 0
    print('indexed by google done')
```

```
#get webpage_rank
webpage_rank = 0
try:
    domain = '{uri.netloc}'.format(uri=urlparse(urls))
    domain = domain.replace("www.", "")
    ENDPOINT = 'https://data.similarweb.com/api/v1/data?domain=' + domain
    resp = get(ENDPOINT)
    if resp.status_code == 200:
        webpage_rank = resp.json()['GlobalRank']['Rank']
    else:
        webpage_rank = 0  # web site not exist anymore
except:
    webpage_rank = 0
print('get rank')
```

These code are used to check if the web still can be indexed by google, and what's its rank.

```
#get ports from shodan
try:
    ipinfo = api.host(page_ip)
    open_ports_number = len(ipinfo['ports'])
except:
    open_ports_number = 0
print('get ports')
```

The above code is used to catch the open ports of the given website. As its shown, in this part we use shodan api to achieve it.

```
try:
    for i in request_info:
        if 'Referer' in i['request']['request']['headers']:
            referers.append(i['request']['request']['headers']['Referer'])
            count_referers = len(set(referers))
except:
    count_referers = 0

try:
    for i in request_info:
        if 'method' in i['request']['request']:
            total_request = total_request + 1
            if i['request']['request']['method'] == 'GET':
                GET_request = GET_request + 1
            if i['request']['request']['method'] == 'POST':
                POST_request = POST_request + 1
            if i['request']['request']['method'] == 'HEAD':
                HEAD_request = HEAD_request + 1
            if i['request']['request']['method'] == 'PUT':
                PUT_request = PUT_request + 1
            if i['request']['request']['method'] == 'DELETE':
                DELETE_request = DELETE_request + 1
            if i['request']['request']['method'] == 'CONNECT':
                CONNECT_request = CONNECT_request + 1
            if i['request']['request']['method'] == 'OPTIONS':
                OPTIONS_request = OPTIONS_request + 1
except:
    GET_request = 0
    POST_request = 0
    HEAD_request = 0
    PUT_request = 0
    DELETE_request = 0
    CONNECT_request = 0
    OPTIONS_request = 0
```

These codes are used to get the frequency of request method. Clearly, there are seven different request, but most of the web only contain ‘Get’ and ‘Post’.

3.2 Usage

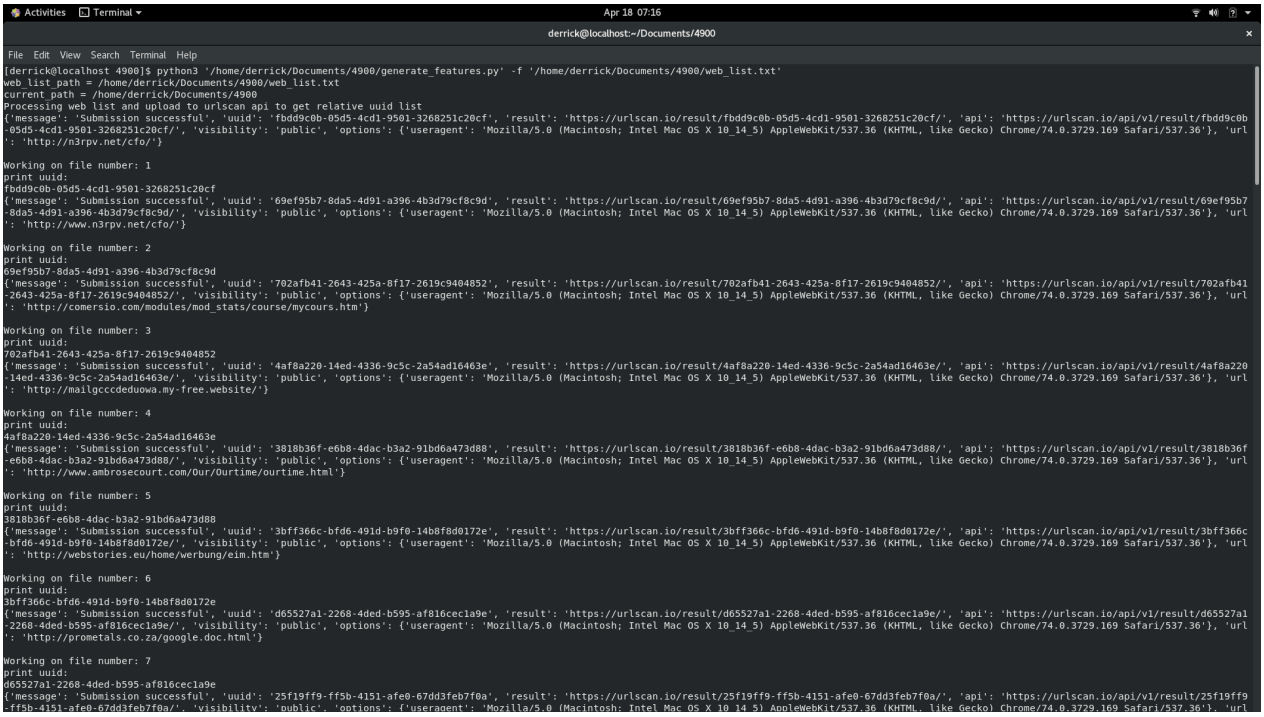


Figure 1.

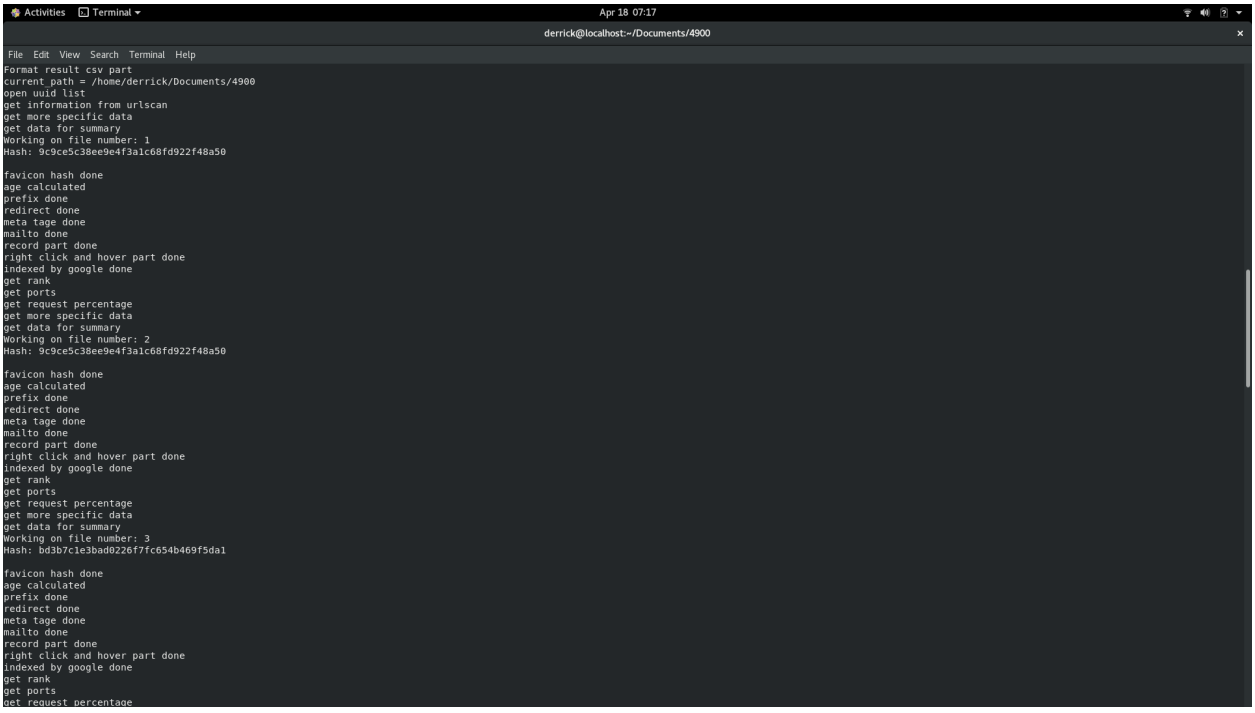


Figure 2

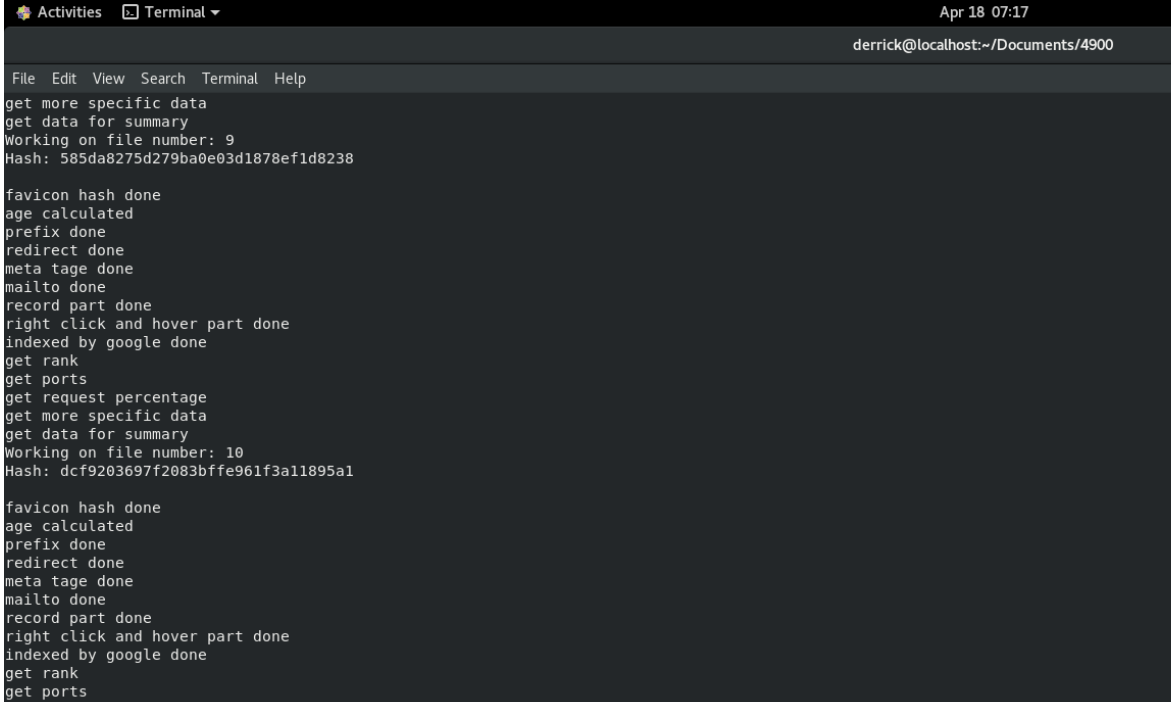


Figure 3

The above figure 1 to 3 are the usage of the script and what is collected in every step.

3.2. AI training

As mentioned above, we use scikit-learn to do the training part because of its simplicity of encapsulating.

```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
import pandas as pd
from sklearn import datasets, linear_model
```

These are import packets, clearly, we import matplotlib, numpy, pandas and sklearn.

```
data = pd.read_csv('/home/derrick/Documents/4900/dataset2.csv')
```

```
data.head()
```

	url	ip_address	Country	server	ptr	asn
0	http://n3rpv.net/cfo/	50.62.224.1	US	Apache	p3nlhg474c1474.shr.prod.phx3.secureserver.net	AS26496
1	http://www.n3rpv.net/cfo/	50.62.224.1	US	Apache	p3nlhg474c1474.shr.prod.phx3.secureserver.net	AS26496
2	http://comersio.com/modules/mod_stats/course/m...	173.201.246.143	US	Apache	ip-173-201-246-143.ip.secureserver.net	AS26496
3	https://mailgcccdeduowa.my-free.website/	2606:4700::6812:828e	US	cloudflare	NaN	AS13335
4	http://www.ambrosecourt.com/Our/Ourtime/ourtim...	52.73.252.105	US	nginx	ec2-52-73-252-105.compute-1.amazonaws.com	AS14618

5 rows × 49 columns

This step is to load our csv file and show top 5 lines of the file.

```
data['Country'] = pd.factorize(data['Country'])[0].astype(np.uint16)
data['server'] = pd.factorize(data['server'])[0].astype(np.uint16)
data['asn'] = pd.factorize(data['asn'])[0].astype(np.uint16)
data['protocol'] = pd.factorize(data['protocol'])[0].astype(np.uint16)
data['MX'] = pd.factorize(data['MX'])[0].astype(np.uint16)
data['NS'] = pd.factorize(data['NS'])[0].astype(np.uint16)
data['SOA'] = pd.factorize(data['SOA'])[0].astype(np.uint16)
```

```
X = data[['Country', 'server', 'asn', 'malicious', 'ads_blocked', 'totalLinks', 'urlScore' ,'secureRequests',
          'securePercentage', 'IPv6Percentage', 'certificate', 'domain_age', 'contain_prefix', 'url_length',
          'dot_number', 'validatity_period_time_stamp', 'is_redirect', 'iframe_number', 'mailtos_number',
          'a_tag_number', 'domain_in_url', 'protocol', 'MX', 'NS', 'SOA', 'right_disable', 'hover_change_status_bar',
          'indexed_by_google', 'webpage_rank', 'a_tag_url_number_percentage', 'meta_link_percentage',
          'contain_popup_window', 'open_ports_number', 'referers', 'GET_request_percentage', 'POST_request_percentage',
          'HEAD_request_percentage', 'PUT_request_percentage', 'DELETE_request_percentage',
          'CONNECT_request_percentage', 'OPTIONS_request_percentage']]
```

This step is used for two purposes. One of the reason is that, country, server, asn, protocol, MX, NS and SOA, all of them are string. However AI model cannot read string directly so we need to trans them to the AI model. The other one is that we need to tell the AI model which prosperities should be used to train.

```
y = data[['Phishing']]
```

This code is to tell the AI model that the phishing property is result what we need.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

In there we separate the dataset into two parts. One is used to train, another is used to test.

```
from sklearn.linear_model import LinearRegression
linreg = LinearRegression()
linreg.fit(X_train, y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False
```

Now are using scikit-learn to mimic our problem.

```
y_pred = linreg.predict(X_test)
from sklearn import metrics
print ("MSE:",metrics.mean_squared_error(y_test, y_pred))
print ("RMSE:",np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
MSE: 0.1054376622744044
RMSE: 0.32471166020702796
```

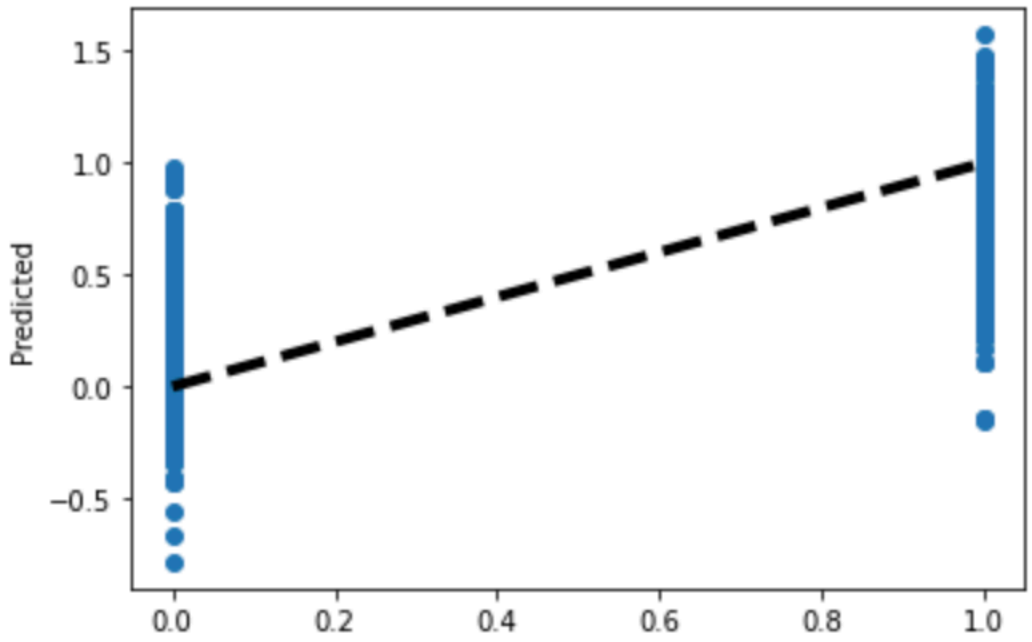
Now we print Mean Square Error (MSE) and Root Mean Squared Error (RMSE) to determine if the model works good.

```
X = data[['AT', 'V', 'AP', 'RH']]
y = data[['PE']]
from sklearn.model_selection import cross_val_predict
predicted = cross_val_predict(linreg, X, y, cv=10)
# 用scikit-learn计算MSE
print "MSE:",metrics.mean_squared_error(y, predicted)
# 用scikit-learn计算RMSE
print "RMSE:",np.sqrt(metrics.mean_squared_error(y, predicted))
```

This is the cross-validation, which is used to continuously optimize the model.

```
fig, ax = plt.subplots()
ax.scatter(y, predicted)
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_ylabel('Predicted')
plt.show()
```

We use this code to generate the training result.



This is the result of the training. Clearly the picture is composed by two parts. One is one 0.0 part another is is 1.0 part. The reason is that we only have phishing and not phishing websites, in this case the result can only be yes or not so the graph has two lines.

Reference

[1]: https://en.wikipedia.org/wiki/TensorFlow

[2]: https://data-flair.training/blogs/tensorflow-pros-and-cons/

[3]: https://en.wikipedia.org/wiki/Artificial_neural_network

[4]: https://en.wikipedia.org/wiki/Artificial_neuron

[5]: https://adventuresinmachinelearning.com/neural-networks-tutorial/#what-are-anns