International Institute of Information Technology Hyderabad

Statistical Methods in Artificial Intelligence

Malicious URL Detection

- Archit Kumar (2018201051)
- Aman Agrawal (2018201006),
- Souparna Das (2018201010),
- Rajneesh Singhatiya (2018202018)

Literature : https://pdfs.semanticscholar.org/93a7/06c58d6e827af2a82a518b4af6b3b53f996d.pdf

Github Repo :
Old Repo - https://github.com/agarwal29796/fraud_page_detection
Final Repo - https://github.com/aman-agrawal/Malicious-URL-detection

Abstract

Malicious Web sites are a cornerstone of Internet criminal activities. They host a variety of unwanted content ranging from spam-advertised products, to phishing sites, to dangerous "drive-by" exploits that infect a visitor's machine with malware. As a result, there has been broad interest in developing systems to prevent the end user from visiting such sites.

Existing approaches to detecting malicious websites can be classified into two categories:

The **static approach** aims to detect malicious websites by analyzing their URLs or their contents . This approach is very efficient and thus can scale up to deal with the huge population of websites in cyberspace. This approach however has trouble coping with sophisticated attacks that include obfuscation and thus can cause high false-negative rates by classifying malicious websites as benign ones.

The **dynamic approach** aims to detect malicious websites by analyzing their run-time behavior using Client Honeypots or similar systems . Assuming the underlying detection is competent, this approach is very effective. This approach however, is inefficient because it runs or emulates the browser and possibly the operating system . As a consequence, this approach cannot scale up to deal with the large number of websites in cyberspace.

Datasets used in the project :

1. https://www.kaggle.com/antonyj453/urldataset#data.csv

This data set only contains web url and corresponding label (good or bad) . Therefor we have designed our features set .

The main reason to write our own code to find features was to capture the liveliness of any web page. Because a web page might go from malicious to benign status over time and vice versa.

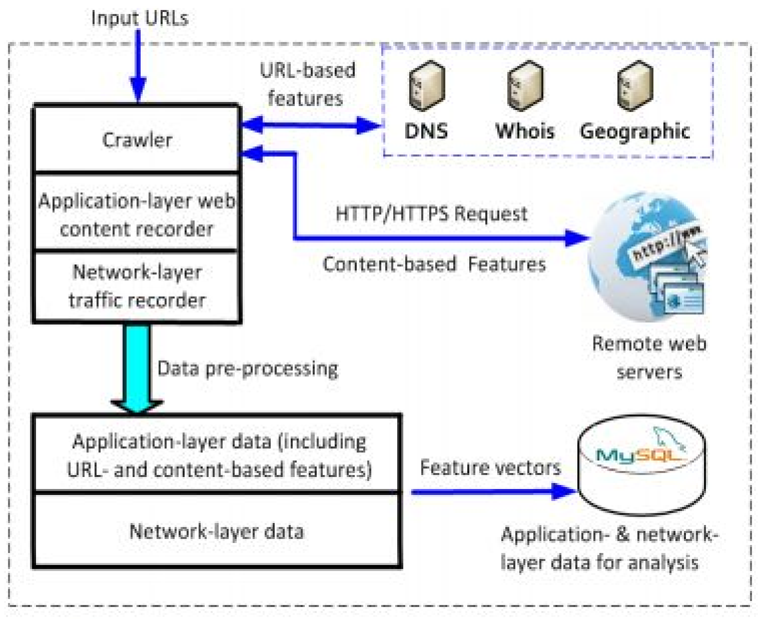
One of the main issue is to be sure that the labelling of training dataset is correct .
To be resolve this issue we can cross check labelling of our training dataset using third party apis.

2. Google drive link of our dataset :

https://drive.google.com/drive/folders/1ntCEfZbnWEaCdk9ro9F-1hisBviAOvU1?usp=sharing

Data Collection, Pre-Processing and Feature Definitions :

At a high level, the data collection system is centered on a crawler. The crawler takes a list of URLs as input, automatically fetches the website contents by launching HTTP requests and tracks the redirects that are identified from the website contents (elaborated below). The crawler also uses the URLs, including the input URL and the detected redirection URLs, to query the DNS, Whois, and Geographic services. This collects information about the registration dates of websites and the geographic locations of the URL owners/registrants.


Input URLs
URL-based features
DNS
Whois
Geographic
Crawler
Application-layer web content recorder
Network-layer traffic recorder
HTTP/HTTPS Request
Content-based Features
Remote web servers
Data pre-processing
Application-layer data (including URL- and content-based features)
Network-layer data
Feature vectors
MySQL
Application- & network-layer data for analysis

The input URLs may consist of malicious and benign websites. A URL is malicious if the corresponding website content is malicious or any of its redirects leads to a URL that corresponds to malicious content; otherwise, it is benign.

Each input URL has an associated application-layer raw feature vector. The features record information such as HTTP header fields, information returned by DNS, Whois and Geographic services, information about JavaScript functions that are called in the JavaScript code embedded into the website content, and information about redirects (e.g., redirection method, whether or not a redirect points to a different domain, and the number of redirection hops).

Data Description :

1. **Url_length** : In order to make malicious URLs hard to blacklist, malicious URLs often include automatically and dynamically generated long random character strings.
data showed that the average length of benign URLs is 18.23 characters, whereas the average length of malicious URLs is 25.11 characters.

2. **special_character_count** :
This is the number of special characters (e.g., ?, -, _, =, %) that appear in a URL. Our data showed that benign URLs used on average 2.93 special characters, whereas malicious URLs used on average 3.36 special characters.

3. **having_ip_address** : It tells whether url contains ip address or not. Some websites use IP addresses instead of domain names in the URL because the IP addresses represent the compromised computers that actually do not have registered domain names.

4. **dns_response_time** : This is the response time of DNS servers. Benign URLs often have longer lifetimes and their domain names are more likely cached at local DNS servers. As a result, the average value of this feature may be shorter for benign URLs.

5. **handshake_time** : handshake time between client and server. Malicious urls generally have large handshake time.

6. **data_time** : time to fetch data from url.

7. **Number_of_redirect**. This is the total number of redirects embedded into an input URL.This feature is unique at the application layer because it cannot be precisely obtained at the network layer, which cannot tell a redirect from a normal link.

8. **Data length** : This feature checks the consistency between the HTTPHeader_content_Length feature value (i.e., the value of the content length field in HTTP header) and the actual length of web content. It is relevant because the content length field could be a negative number, which may cause buffer overflow attacks.

9. **redirect** : tells whether this url redirected to some other url or not

10. **protocol** : servers that are heavy on dynamic content tend to be impacted less by HTTPS because the time spent encrypting (SSL-overhead) is insignificant compared to content generation time. Servers that are heavy on serving a fairly small set of static pages that can easily be cached in memory suffer from a much higher overhead.

11. **http_status_code** :
   200 Ok
   203 Non-Authoritative Information
   204 No Content
   307 Temporary Redirect
   400 Bad Request
   401 Unauthorized
   402 Payment Required
   403 Forbidden
   404 Not Found

12. **server_name** : tells who is hosting this web .

13. **scripts_count** : no. of javascripts. JavaScript functions are often used by attackers to obfuscate their code and bypass static analysis. For example, eval() can be used to dynamically execute a long string at runtime, where the string can be the concatenation of many dynamic pieces of obfuscated substrings at runtime; this makes the obfuscated substrings hard to detect by static analysis. Script plays a very important role in drive-by download attack.

14. **iframe_count** : no. of iframes in web . If any iframe contains malicious code, the URL is malicious. A small size iframe is even more harmful because it imports malicious content that is invisible to the users.

15. **external_link_count** : no. of external links. This feature can be indicative of malicious URLs because external URLs are often abused by attackers to import malicious content to hacked URLs.

16. **total_strings_count** : no. of strings in javascript .
17. **average_string_length**
18. **max_len_strings**
19. **strings_above_avglength**
-Because attackers try to encode malicious script code into a string and then use it to execute in client browser, these(16,17,18,19) features can be indicative of malicious URLs.

20. **whois_country:** it is a categorical variable, its values are the countries we got from the server response (specifically, our script used the API of Whois).

21. **whois_city:** it is a categorical variable, its values are the states we got from the server response (specifically, our script used the API of Whois).

22. **whois_creation_date:** Whois provides the server registration date, so, this variable has date values with format DD/MM/YYYY HH:MM

23. **whois_last_updated:** Through the Whois we got the last update date from the server analyzed .

24. **Whois_expiry_date** : It is expiry date of the registration in the whois database.

What Other features we can use :

These features were not possible to extract due to system requirements and To extract these features we need some safe system like : honeypots.
Unluckily these were also the features which showed highest importance in the dataset 2.(available on kaggle)

Features based on crawler-server communication (network layer features):

1. **App_bytes.** This is the number of Bytes of the application-layer data sent by the crawler to the remote web server, not including the data sent to the DNS servers. Malicious URLs often cause the crawler to initiate multiple requests to remote servers, such as multiple redirections, iframes, and external links to other domain names. The average App_bytes is 36818 bytes for malicious websites and 53959 bytes for benign websites.
2. **Source_app_packets.** This is the number of packets send by the crawler to remote servers .
3. **REMOTE_APP_PACKETS:** packets received from the server .
4. **TCP_CONVERSATION_EXCHANGE:** This variable is the number of TCP packets exchanged between the server and our honeypot client .
5. **APP_PACKETS:** this is the total number of IP packets generated during the communication between the honeypot and the server .

Page-Based Features

Page-Based Features are using information about pages which are calculated reputation ranking services. Some of these features give information about how much reliable a website is. Some of Page-Based Features are given below.

Global Pagerank
Country Pagerank
Position at the Alexa Top 1 Million Site
Some Page-Based Features give us information about user activity on target site. Some of these features are given below. Obtaining these types of features is not easy. There are some paid services for obtaining these types of features.

Estimated Number of Visits for the domain on a daily, weekly, or monthly basis
Average Pageviews per visit
Average Visit Duration
Web traffic share per country
Count of reference from Social Networks to the given domain
Category of the domain
Similar websites etc.

Malicious Javascript : Malicious javascript often utilizes obfuscation to hide known exploits and prevent rule-based or regular expression (regex)-based anti-malware software from detecting the attack. The complexity of obfuscation techniques has increased, raises the resources necessary to deobfuscate the attacks. obfuscated JavaScript based redirection, is still a challenging open problem .

Data Cleaning :

Handshake_time : for those urls for which we are unable to fetch this. We put 0 and for some uncatched errors we remove these urls.

Data_time : for those urls for which we are unable to fetch this. We put 0 and for some uncatched errors we remove these urls.

Data_length : for those urls for which we are unable to fetch this. We put 0 and for some uncatched errors we remove these urls.

Server_name : for those urls for which we are unable to fetch this. We introduce another category for them.

Redirect : for those urls for which we are unable to fetch this. We removed them.

Whois_creation_date : for those urls for which we are unable to fetch this. We introduce another category for them.

Whois_expiry_date : for those urls for which we are unable to fetch this. We introduce another category for them.

Whois_last_updated : for those urls for which we are unable to fetch this. We introduce another category for them.

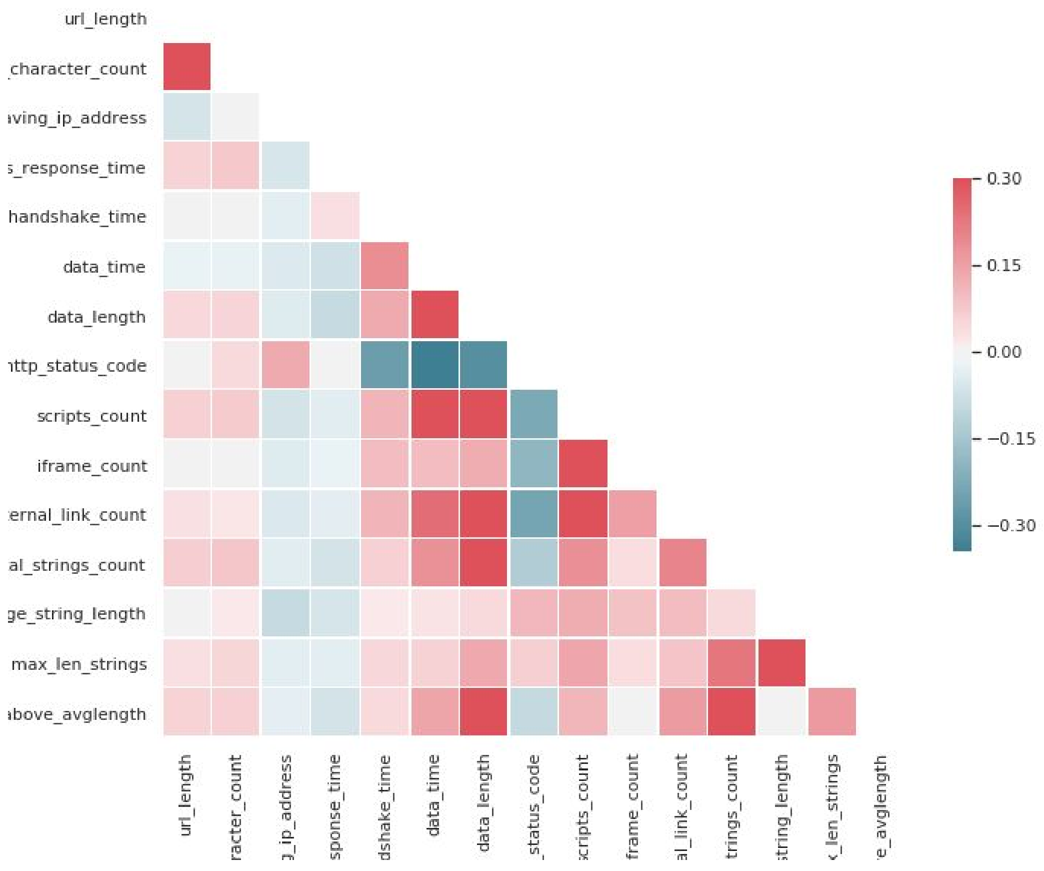
Whois_country : for those urls for which we are unable to fetch this. We introduce another category for them.

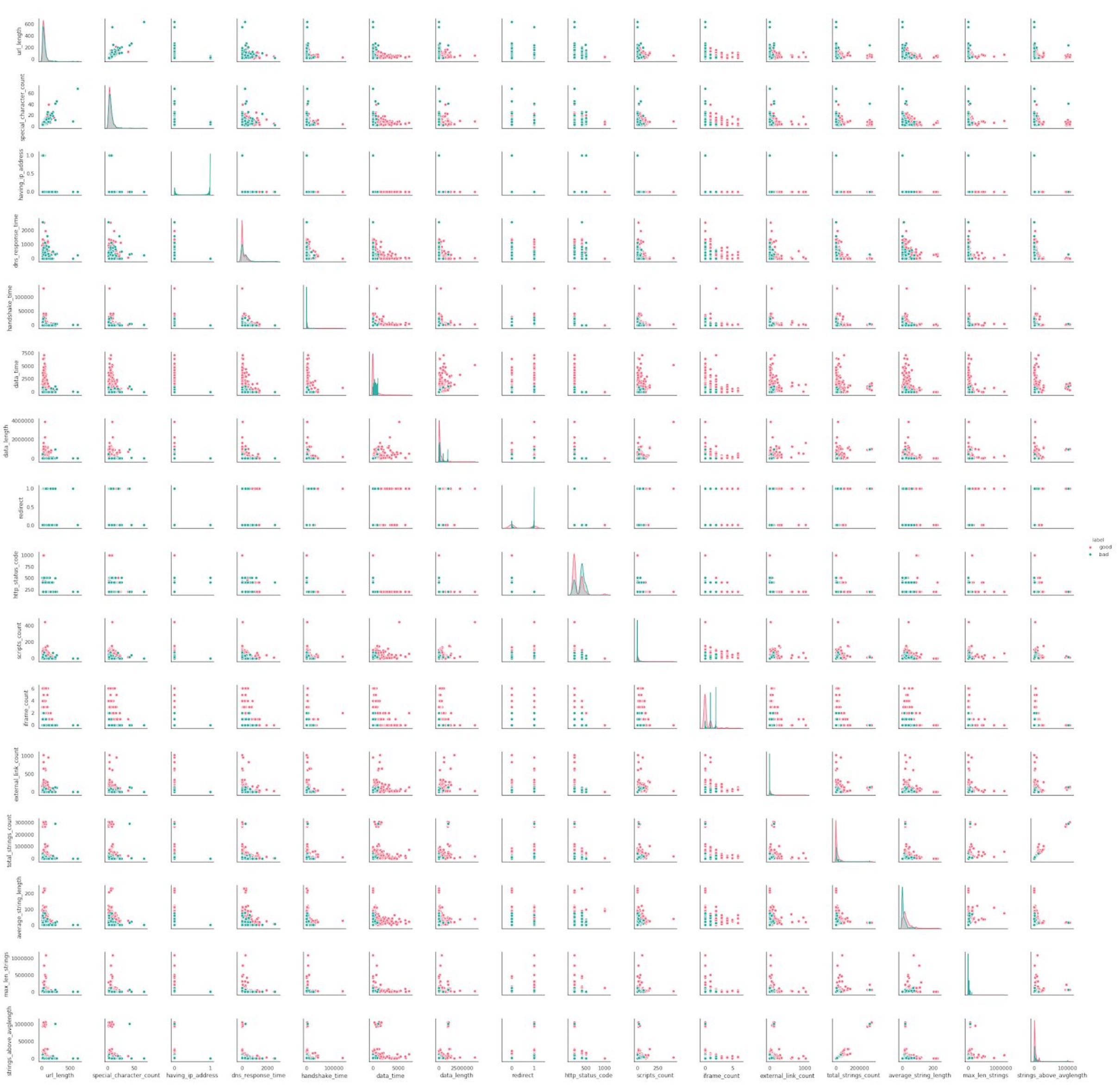
Whois_city : for those urls for which we are unable to fetch this. We introduce another category for them

Data Analysis :

Important observations :

1. Benign websites tend to have less url length and special_character_count and malicious websites tend to have large url length and special_character_count.
2. Benign websites tend to have large data_time and data_length
3. Benign websites do not contain ip addresses.
4. Benign websites tend to have 200 as http_response_request and malicious websites tend to have 404,502,503.
5. When there are zero scripts_count, there are a lot of malignant entries(134) and they decrease with increase in value of scripts_Count.
6. Most of the bad and good urls have zero iframe_count, total_string_count, average_string_length, strings_above_avglength and external_link_count. Both good and bad decrease with increase in iframe_count, total_string_count, average_string_length, strings_above_avglength and external_link_count.

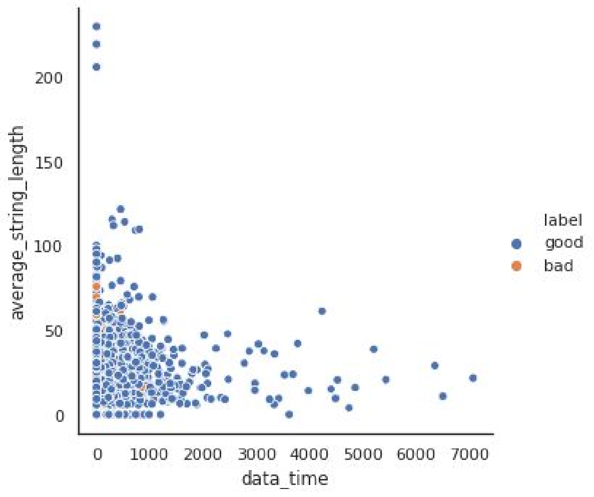

url_length
_character_count
aving_ip_address
s_response_time
handshake_time
data_time
data_length
ttp_status_code
scripts_count
iframe_count
ernal_link_count
al_strings_count
ge_string_length
max_len_strings
above_avglength
url_length
racter_count
g_ip_address
sponse_time
dshake_time
data_time
data_length
_status_code
scripts_count
frame_count
al_link_count
trings_count
string_length
_len_strings
e_avglength
0.30
0.15
0.00
−0.15
−0.30

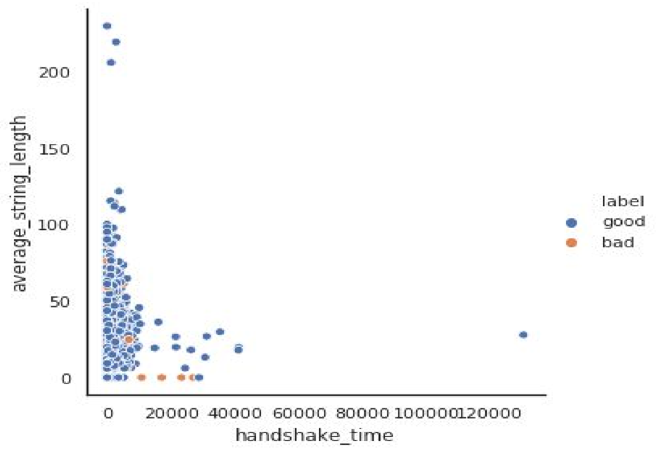

url_length
special_character_count
having_ip_address
dns_response_time
handshake_time
data_time
data_length
redirect
http_status_code
scripts_count
iframe_count
external_link_count
total_strings_count
average_string_length
max_len_strings
strings_above_avglength
label
good
bad

Multivariate Analysis :

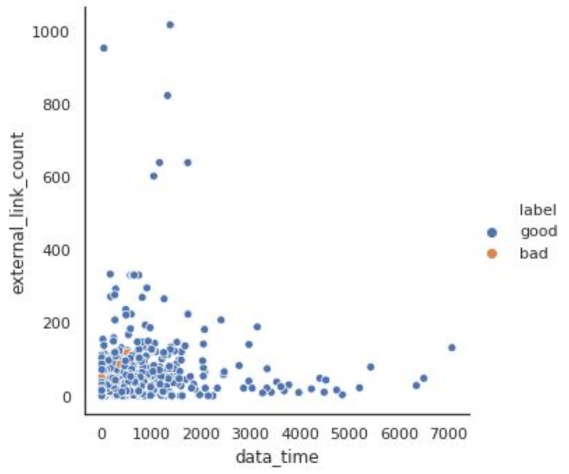
After observing graphs plotted by pairplot and covariance matrix, we went through each one of them and tried to find any relation between any two variables and if there are any patterns developing in the graphs.

There were few interesting scenarios developed in the whole process. In some cases we could mark in which part of graph are good and bad concentrated, and whether they overlap each other or not. Our main target here was to find the dependence of the label on two features simultaneously. Following are some of the interesting graphs plotted in exploration.ipynb file.

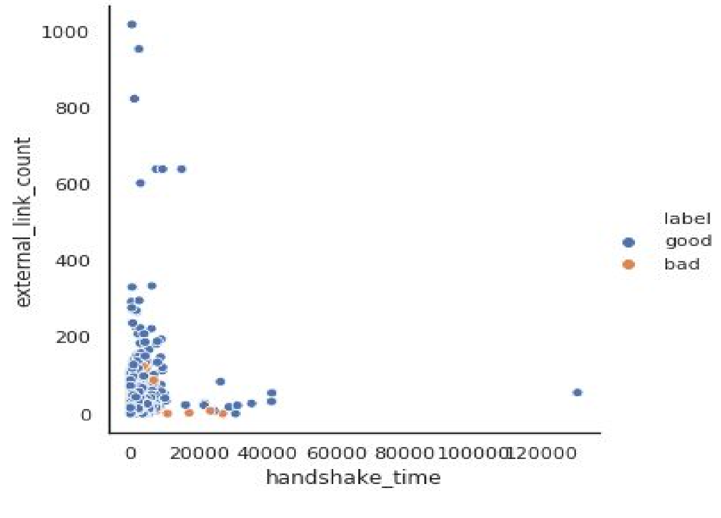




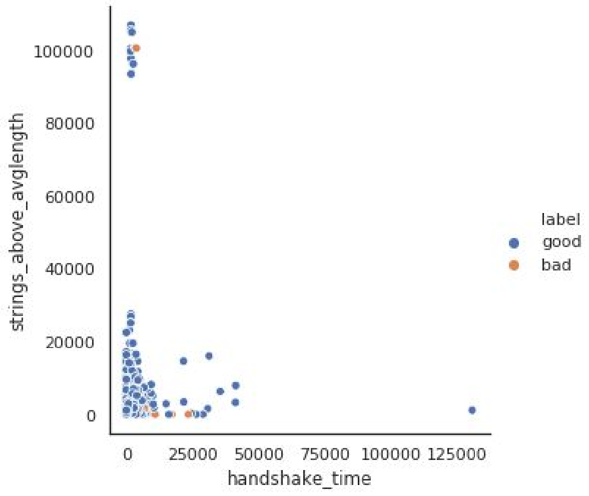
average_string_length
handshake_time
0
50
100
150
200
0
20000
40000
60000
80000
100000
120000
label
good
bad



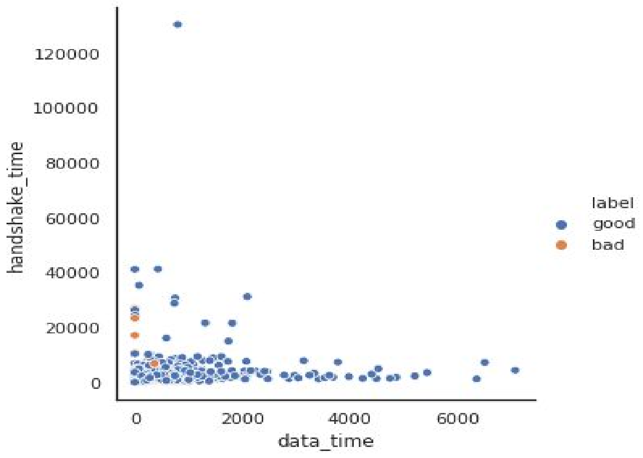
external_link_count
data_time
0
200
400
600
800
1000
0
1000
2000
3000
4000
5000
6000
7000
label
good
bad



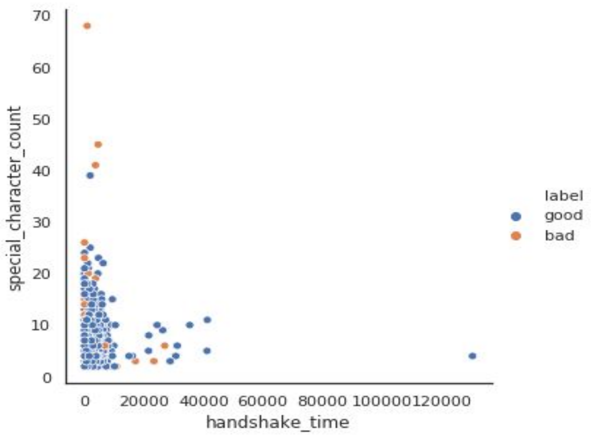
external_link_count
handshake_time
0
200
400
600
800
1000
0
20000
40000
60000
80000
100000
120000
label
good
bad



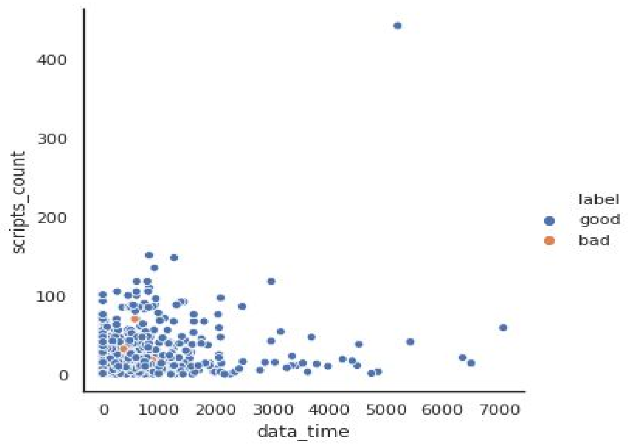
strings_above_avglength
100000
80000
60000
40000
20000
0
0
25000
50000
75000
100000
125000
handshake_time
label
good
bad



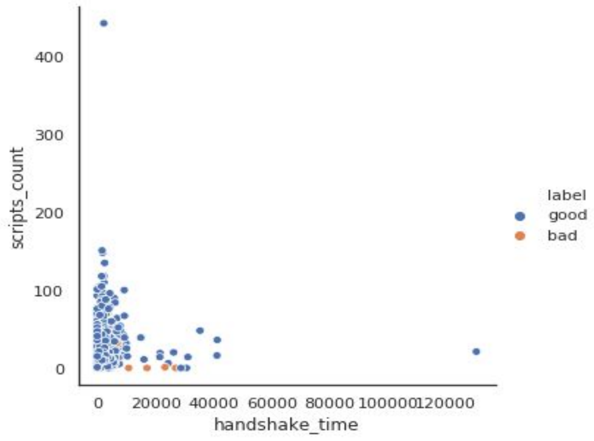
handshake_time
120000
100000
80000
60000
40000
20000
0
0
2000
4000
6000
data_time
label
good
bad



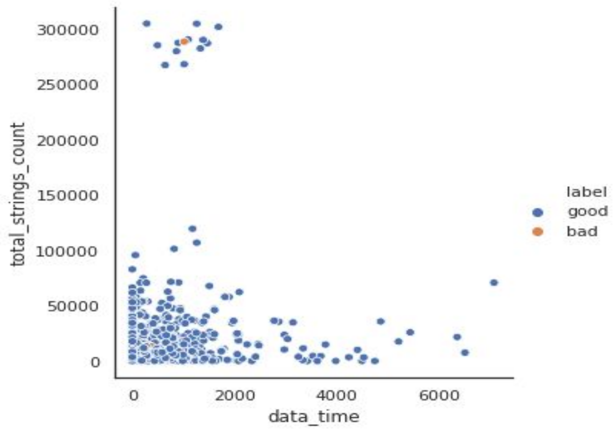
special_character_count
70
60
50
40
30
20
10
0
0
20000
40000
60000
80000
100000
120000
handshake_time
label
good
bad



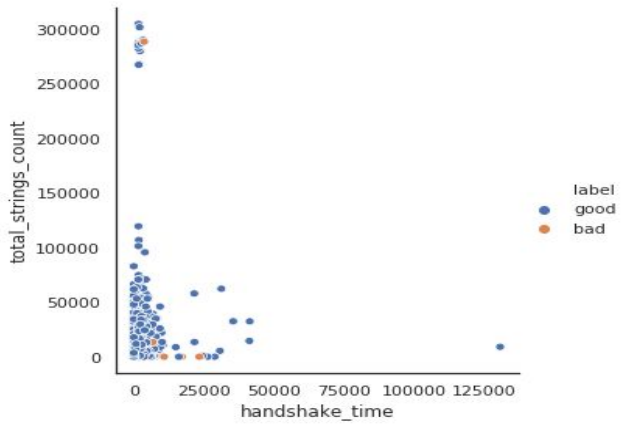
scripts_count
400
300
200
100
0
0
1000
2000
3000
4000
5000
6000
7000
data_time
label
good
bad



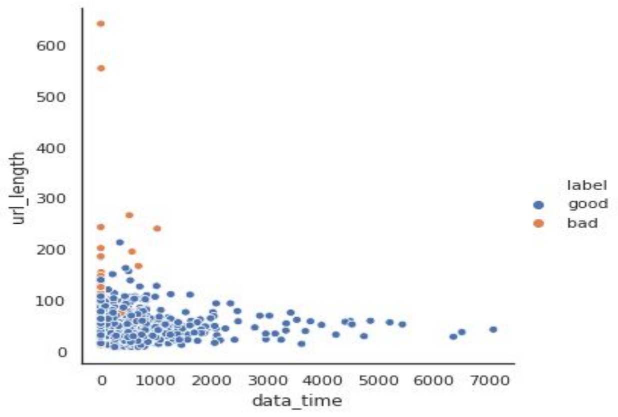
scripts_count
400
300
200
100
0
0
20000
40000
60000
80000
100000
120000
handshake_time
label
good
bad



total_strings_count
300000
250000
200000
150000
100000
50000
0
0
2000
4000
6000
data_time
label
good
bad



total_strings_count
300000
250000
200000
150000
100000
50000
0
0
25000
50000
75000
100000
125000
handshake_time
label
good
bad



url_length
600
500
400
300
200
100
0
0
1000
2000
3000
4000
5000
6000
7000
data_time
label
good
bad

Models used :

1. SVM

We have used Linear and Kernel SVM both for this problem. The dataset is highly imbalanced and it’s a big reason for suffering of some machine learning models.

Linear SVM:
Has performed well than most of the other models, with precision = 0.45, **recall = 0.70**, for the BAD urls. For the GOOD urls the **precision = 0.95** and and **recall = 0.87.** The accuracy on validation set is 85% which is comparable. The AUROC value of Linear SVM is **0.79.** Here is the performance chart for the Linear SVM:

	precision	**recall**	**f1-score**	**support**
BAD URL	**0.45**	**0.70**	**0.55**	**71**
GOOD URL	**0.95**	**0.87**	**0.91**	**457**

We can see that the recall is high for the bad url which we want and this is important as a BAD url should not be misclassified as a good URL.

Kernel SVM:
The precision, recall using Kernel SVM is not on par with the Linear SVM, though its not bad compared to other models as the **AUROC** is **0.71.** The performance on the validation set is given below:

	precision	**recall**	**f1-score**	**support**
BAD URL	**0.37**	**0.56**	**0.45**	**71**
GOOD URL	**0.93**	**0.85**	**0.89**	**45**

Linear SVM has a better recall than kernel SVM.

2. Neural Network

Parameters :

Optimizer : 'lbfgs' is an optimizer in the family of quasi-Newton methods.

Layers Size : (50, 2)

	Precision	Recall	F1 score	Support
Bad url	0.41	0.43	0.42	70
Good url	0.92	0.92	0.92	531

3. Logistic Regression

Logistic Regression has performed well for this problem. But there is catch logistic regression tends to be highly biased for imbalanced dataset, we had to use class weights to make it work well. Here is the performance detail of logistic regression on validation set:

	precision	**recall**	**f1-score**	**support**
BAD URL	**0.35**	**0.83**	**0.50**	**69**
GOOD URL	**0.97**	**0.77**	**0.86**	**459**

As we can see, the logistic regression gives a **recall=0.83** for BAD urls and it by far the best recall we have got from any other model. The AUROC value of Logistic is 0.80

4. Random forest

Using Ensemble models often boosts the performance, but in our case **RANDOM FOREST** has not performed as well as Logistic Regression does. The reason for this is the dataset is highly **imbalanced** and it affects a classifier like random forest as a **pure leaf node is often impossible to get** in most cases of imbalanced data set. We have used **grid search** and

k-fold cross-validation with **6336** combination and got the best possible random forest estimator which performance on unseen data is as follows:

Best Estimator Parameter details:

Number of trees in random forest :	459
Min samples split:	3
Min samples leaf':	3
Max features :	sqrt
max_depth:	118
bootstrap:	True

Best Estimator performance on unseen data after GRID SEARCH:

	precision	**recall**	**f1-score**	**support**
BAD URL	**0.45**	**0.77**	**0.57**	**65**
GOOD URL	**0.96**	**0.87**	**0.91**	**63**

Contribution :

Archit Agrawal - Feature Extraction and Data Modelling , Neural Network Training
Aman Agrawal - Feature Extraction and Data Modelling , SVM Training
Souparna Das - Model Training & Testing and Model Analysis and Models Improvement
Rajneesh Singhatiya - URL based feature extraction ,Data Analysis and Logistic Training