Malicious Web Page Prediction

Nadine Safwat - 900212508

Link to scripts: 🗀 Machine Learning Project

INTRODUCTION

To recap, this project is attempting to create a malicious URL detection system using machine learning models. To begin I gathered research on existing systems that use the same technique and chose an appropriate dataset to train my models. Then I preprocessed the data so that it is suitable and allows the models to train and predict fairly. The next step was to test all the models I could to see the scores they would yield so that I could choose the best to use for the detection system. In the end Random Forest was the best choice and the final steps include fine tuning the hyper parameters and integrating it into an easy to use system.

MODEL IMPLEMENTATION

To implement my model I used the Sklearn RandomForestClassifier. To ensure that the model was not overfitting on the provided datasets and that the model could predict urls retrieved from a third party I found 20 malicious and 20 benign urls, preprocessed them and used them as my test set. After applying Gridsearch and doing some trial and errors by hand the final parameters that i defined in my classifier are:

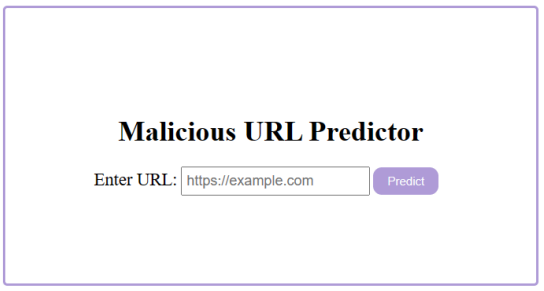
- class_weight={0: 20, 1: 1}:
  - This is done so that the model will give 20 times more importance to class 0 than to class 1 during training to counteract the effect of my imbalanced classes.
- max_depth=30:
  - the maximum depth of each tree will be 30. This parameter sped up the training by 10 minutes however without reducing the metrics

No other parameter increased the metrics of both classes instead giving one class very good scores and the other metrics down in the 0.2s. The metrics for the new test set predicted with the parameters chose above are:

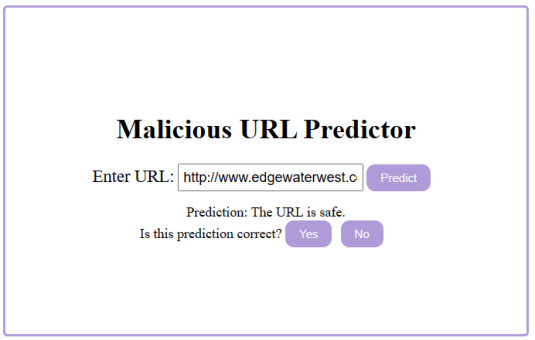
```
Classification Report:
              precision    recall  f1-score   support

           0       0.52      0.55      0.54        20
           1       0.53      0.50      0.51        20

    accuracy                           0.53        40
   macro avg       0.53      0.53      0.52        40
weighted avg       0.53      0.53      0.52        40
```

CLIENT IMPLEMENTATION

To create my user interface I used HTML, CSS and Javascript to create a simple easy to understand page. The application will first open on the main page that will ask the user to input a URL to predict and a button called predict that they can press.



Once the user enters their url and presses the predict button the page will display the prediction of the url and the User will be prompted to provide feedback



If the user provides feedback, a message will be displayed confirming their feedback and thanking them.

SERVER IMPLEMENTATION:

Database:

I created a database on MySQL Workbench to store the train set so that the server can read the data when it needs to retrain or so that the URLs that have feedback provided can be entered into the database. The Database contains only one table called data_to_view that contains the rows:

Url (PK),url_numOf_digits, url_entropy, url_len, ip_add, geo_loc, tld, who_is, https, js_len, label

I decided to keep the geo_loc and tld columns categorical and one hot encode them after the database is read due to memory issues as well as ease of tracking the URL information. For the same reason the ip_add is stored in its dot format and is transformed after reading from the database.

Preprocessing:

Realistically, the only input that can be asked from the user is the URL of the webpage so i have multiple function to get all the features that i require for the prediction:

- url_numOf_digits
    - Found by analyzing the url string using: sum(c.isdigit() for c in s)
- url_entropy:
    - The same function that was used in MS2 is called to calculate the entropy of the URL
- url_len:
    - Found by analyzing the url string using: len(url)
- ip_add:
    - To get the IP address there are two steps. First I use the requests and urlib libraries to find the ip address of the webpage, then this ip address is sent to the function transform_ip that will take the dot format and will return its decimal equivalent.
- https:
    - Found by analyzing the url string using: 1 if url.startswith('https') else 0
- js_len:
    - Using the requests and BeautifulSoup library the function will visit the webpage and retrieve the javascript, then the length of all the tags are summed to get the javascript length
- geo_loc:
    - Using the requests and urlib libraries the web page is visited and the hostname retrieved. Then using an API call to IPinfo the geographical location is retrieved.
- tld:
    - To get the top level domain the class tldextract is used and the suffix is sent as tld

After all these features are extracted and stored in a dataframe, geo_loc and tld must be one hot encoded. This is done by reading the trained one hot encoder from a pickle file and applying .transform on the two columns. Now, the data is ready to be predicted.

APIs

To facilitate the communication between the client and the server I used the Flask and requests library in Python to define two POST APIs for the client to communicate with the server.

The first API will retrieve the inputted url from the client, preprocess it as described, and predict the label using the trained model that is stored in a pickle file. It will then send the prediction back to the client so that it can be displayed to the user.

The second API will retrieve the inputted url, the predicted label and the users feedback. Using the feedback the predicted label will be handled accordingly and the url will be processed. Then this data will be stored in the database so that it can be used to retrain the model. Once the data is saved into the database a message is sent back to the client that says “URL Saved”

Retraining:

To retrain the model, the server will first query the database and to get the number of rows as the previous number of rows. Then, inside an infinite loop, this query will be done again every minute to get the current number of rows. To check if we need to retrain the server will check if this number has increased by a chosen number like 1k (For the demo i chose 2) and will retrain the model. This retraining is done by reading all the rows in the database, which will include the new predictions, preprocess the required columns, and refit the data on this set. It will then overwrite the old pickle file with the retrained model. This process happens continuously in the background and is not dependent on the movements of the client.