July 28, 2025

Tsion Gebissa

C964: Computer Science Capstone

Task 2 parts A, B, C, and D

Part A: Letter of Transmittal

July 28, 2025

Attn: Alex Miller

CrBlock Solutions

1234 State Drive

Seattle, WA 98897

Re: Malicious URL Detection Using Machine Learning

Dear Mr. Miller,

With the growing public trust in cryptocurrency, CrBlock is projected to see a rapid increase in its customer base. This is a good opportunity to ensure the company's security protocols are reliable to protect customer data from cybersecurity threats. One of the main ways cybercriminals manage their attacks is by relying on malicious links. These links can often be found in emails, website advertisements, and documents. These links are used by attackers to compromise systems, sometimes leading to shutdowns or exposure of customer data. While CrBlock has training in place to help employees identify the many social engineering tactics used by malicious actors, it currently lacks an automated system to detect and block malicious links.

The solution I am proposing will involve the development of an internal machine learning tool to detect and flag malicious links. The deliverables will include a trained and tested machine learning model and a simple user interface that enables testing URLs for potential threats, including phishing, malware, and defacement. Upon budget approval from your leadership team, this model will be deployed company-wide and embedded into CrBlocks's system.

The solution will focus on identifying phishing, malware, and defacement URLs. The phishing technique uses social engineering to trick users into giving out sensitive information. Malware is often delivered through malicious URLs and can compromise devices or lead to data theft. Defacement attacks commonly alter website content and may be used to damage a company's reputation.

This solution will benefit CrBlock as it will provide a proactive measure to protect user data. The company-wide measure will help prevent data breaches and safeguard user information. This will ensure user trust and that the company's reputation is maintained. It will also ensure that legal compliance is met. This prevention method could also potentially reduce costs associated with security incidents and minimize manual cybersecurity tasks.

Initial development and validation will be completed using open-source tools, keeping initial costs low. Integrating the model with the company's current system will require an estimated $50,000 to $75,000 for initial implementation, with an annual recurring cost of approximately $15,000 for data storage, model revisions, and cloud computing resources. The estimated implementation timeline is 6-8 weeks.

The project will not use customer data and will be developed in an isolated environment when handling malicious URLs. A publicly available dataset containing malicious links will be used for model training and testing. Different machine learning models will be tested and measured for performance.

The cybersecurity department leading this project includes engineers and computer scientists with years of experience using modern machine learning tools. They have led several successful large-scale security projects and will bring relevant experience.

Thank you for considering this proposal, and I look forward to discussing any questions or concerns you may have.

Sincerely,

Tsion Gebissa

Part B: Project Proposal Plan

Project Summary

CrBlock currently does not have an automated system to detect malicious URLs. The company relies instead on remedial strategies to address threats. This increases the risk of cyberattacks that could expose customer information, including cryptocurrency transactions. To address the gap, this project proposes a machine learning based system that will scan, detect, and classify URLS before they can impact the network.

This machine learning model will be embedded into the company's system to detect malicious URLs that enter the system. The machine will predict the safety of URL links and classify them into phishing, malware, and defacement. Identified malicious links will then be added to a blacklist database, and the internal IT team will manage this database to ensure ongoing protection is maintained.

The deliverables for this project include:

- A trained machine learning model that can classify URLs as benign, phishing, malware, or defacement.
- A Jupyter Notebook application that demonstrates the model in action. This also provides a blueprint for future feature refinement that could be added by taking advantage of the vast list of malicious links that will be added to the blacklist database. This will help recognize patterns and identify features that could be extracted and applied to the model to make it more robust.
- A user guide with clear instructions on using the user interface embedded into the Jupyter Notebook. This interface will be used to test the model's functionality before deployment.
- Supporting materials, including charts, metrics, and examples that make the product replicable. These can be used to test the model's behavior and performance in a controlled testing environment.

Data Summary

The dataset for this project will be obtained from a publicly available Kaggle dataset containing approximately 651,191 URLs and their label. These labels are benign, defacement, phishing, and malware. This dataset was selected because of the vast variety of sources the author used. The dataset also meets the needs of this project as it contains the different malicious URL classifications that this project is targeting to identify.

The dataset will be downloaded as a raw CSV file and imported into Jupyter Notebook for exploration and pre-processing. The exploration phase will play a crucial part in identifying features to be extracted. These features are a core part of the training phase, as the model will need meaningful numeric data to learn URL patterns.

Using Python libraries, the data set will be checked for missing data. Those rows will be deleted to maintain the dataset's structure if missing data is found. A check will also be performed for duplicate data. Duplicate data can lead to inaccurate data analysis, impacting the model's actual performance. Any identified duplicate data will be truncated.

Once the data is thoroughly cleaned, visualizations will be created to help understand the data characteristics and any issues that might need to be addressed. These visualizations will include an overview of the different classes(labels) and other features to measure for class imbalance and other irregularities that could affect the model's performance.

There are no ethical or legal concerns regarding the dataset. The dataset is publicly available and hosted on a reputable website trusted and used by data scientists and others exploring machine learning and data analysis.

Implementation

The implementation of this project will be guided by the **CRISP-DM**(cross-industry process for data mining) methodology (Smart Vision Europe, n.d.). This methodology is widely used in projects involving data processing. The implementation phases for this project using CRISP-DM are as follows:

Business Understanding:

This stage will focus on communication with the leadership team and various departments within the company to gather an understanding of specific priorities. The leadership team will help outline the company's cybersecurity. Information collected from the security team will help identify any recurring cyberattacks involving malicious URLs to which the company is susceptible by making use of any currently existing blacklist and URL logs. The information collected will help identify the feature extraction decisions for model training, the identification of applicable machine learning algorithms, and the tools needed to complete the project. This phase will deliver a defined project scope, timeline for project completion, and an understanding of resources required.

Data Understanding:

At this project stage, the data set will be collected and analyzed for data structure, type, size, and the relationship between URLs and their classification to determine the data preparation approach. The data will also be manually inspected to ensure that only the two columns(URLs and labels) exist and that there are no outliers. This stage will also help determine how the data can satisfy the requirements gathered during the business understanding stage.

Data Preparation:

At this project stage, any duplicate or missing data will be dropped, and the dataset will be analyzed for features to be extracted. Visualizations such as a bar graph will assist in feature extraction decisions. The prepared dataset will then be divided into train and test sets to support model development and evaluation.

Modeling:

The models will be trained and tested to identify which performs best for malicious URL detection based on their accuracy and metrics from their classification report. This stage aims to select the model that provides the most reliable performance across all categories.

Evaluation:

The selected model will be evaluated during this stage to determine if it meets the goals identified in the business requirement stage. The security team and leadership will review performance results to assess if the model is effective in protecting the company against malicious attacks. If the model is found to be successful, the project will proceed to deployment. If the model is not deemed successful, feedback will be taken to revise the model and enhance how URL features are analyzed to detect malicious activity.

Deployment:

This stage requires collaboration from the development, leadership, cybersecurity, and IT teams to ensure all resources are available for deployment. This stage includes the following:

- Gathering resources for data storage to support the blacklist database of malicious URLs that the model will collect.
- Ensuring network support allows the model to integrate into internal systems, such as email servers, the company website, and other devices.
- Providing detailed documentation to allow for the maintenance of the model.

Timeline

Milestone or deliverable	Project Dependencies	Resources	Start and End Date	Duration
Define project goals	Availability of leadership, security and IT teams	Meetings	07/29/2025 – 07/31/2025	3 days
Data collection and exploration	Defined project goals	Dataset, visualization tools	08/01/2025 – 08/05	5 days
Data Preprocessing and formatting for training	Complete data review	Python, Jupyter Notebook	08/05 – 08/15	11 days
Model training, testing, final model selection and user interface development	Prepared dataset	Scikit learn machine learning libraries, evaluation metrics	08/16 – 08/25	10 days
Evaluation of model success	Model trained and selected, user interface available for use, stakeholders are available to provide feedback	Evaluation metrics and visualizations	08/26 – 09/01	7 days
Deployment	The model has been approved for deployment; and resources and budget have been allocated	Availability of leadership, security and IT teams.	09/02 – 09/15	14 days

Evaluation Plan

At each development stage, verification will be performed to ensure that the outcome of each stage aligns with the product goals. This begins with the business understanding stage. The verification will be the completion of goal identification and the outlining of success criteria.

During the data processing stage, this will include output display to verify proper data loading, identifying missing or duplicate data, and using test URLs to verify that feature extraction methods return expected features.

During model training, verification will ensure that the model is only trained on the training set. This is important as the model needs to be trained on new data it has not seen before to accurately measure how it would perform in a production environment.

Verification of the evaluation stage will include confirming that the correct performance metrics are applied and that the results are based only on the test data. Further verification will ensure that stakeholder feedback is collected before making any deployment decisions.

Deployment verification will include checking that all required documentation is complete, data storage and budget are allocated, a clear outline of deployment steps, such as system integration, is provided, and responsibilities for maintaining the model beyond deployment are defined.

Upon completion of the project, validation will be performed, which will involve evaluating model accuracy, precision, and recall using quantitative measures. In addition to performance validation, the user interface will be tested on multiple devices to confirm that the program is compatible with different operating systems.

Resources & Costs

Resource	Description	Estimated Costs and/or Hours
Software	Python : Version 3.3 – Open Source	$0
	Google Colab : Free cloud-based notebook environment	$0
	Libraries - Scikit-learn, pandas, matplotlip, etc. Open-Source Libraries	$0
Dataset	Malicious URL Dataset	$0
Hardware	Laptop or Computer – Existing resource	$0
Deployment and System Integration	Initial deployment and system integration	$20,000
	Cloud services – Microsoft Azure Machine Learning: Subscription Model - Includes computing resources and data storage	$250 per month
Labor Cost	Software Engineer	$65/hr * 100 hrs = 4500
	Senior Software Engineer & Project Lead	$70/hr * 120 hrs = $8400
	Data Analyst	$60/hr * 80 hrs = $4800
	Machine Learning Expert	$85/hr * 80 hrs = $6800
	Project Lead	$65/hr * 40 hrs = $2600
Total Cost		$47,100 initial cost $250/per month for maintenance

Part C: Application

The application is hosted on Google Colab as a Jupyter Notebook. **All necessary libraries and imports are included in the Notebook and will be automatically downloaded/imported upon running the program**. Once the application is entirely run, the user will be asked to enter a URL for safety prediction at the end of the notebook. The user may use the sample test URLs provided in a drop-down menu or enter their URL to test

For a step-by-step guide, please see the User Guide

- The **Jupyter Notebook** is located at the following link: https://colab.research.google.com/drive/1a4t6rzFe1__GxFr8nJK5yuCtVprbDquw?usp=sharing
- The original dataset can be found at the Kaggle website: https://www.kaggle.com/datasets/sid321axn/malicious-urls-dataset
- The CSV file is uploaded to GitHub and imported into the Jupyter Notebook. This allows the user to run the application without needing to upload the file. The **CSV dataset** can be located at the following link: https://github.com/T-Geb/Malicious-URL-detection-_-ML/blob/main/trimmed_dataset.csv

Part D: Post-implementation Report

Solution Summary

The problem addressed in this project was the need to implement an automated system to detect and classify malicious URLs within CrBlock's internal system. The need was justified by the recent growth the company experienced, the increase in workforce, and the use of distributed systems that make the company susceptible to malicious attacks. The solution presented is applying a machine learning model to a dataset containing labeled data used to train the model. Extensive work was done on data preparation and preprocessing to ensure that relevant features were extracted for model training. The deliverables of this project include a user interface that can be used to make predictions on new URLs to identify their safety. The model predicts whether a URL is benign, defacement, malware, or phishing. This reduces reliance on manual detection and helps build a dataset that can be used for future model refinement and feature engineering to better detect patterns in malicious URLs.

Data Summary

The dataset for this project was sourced from a publicly available dataset on Kaggle titled "Malicious URLs Dataset". This set has two columns labeled 'urls' and 'type'. The 'urls' column contains URLs, and the 'type' column contains the classification types for each URL. These classifications are – benign, defacement, malware, and phishing.

Dataset Issue Identified:

The dataset was downloaded as a CSV file. During inspection, a discussion on Kaggle highlighted that a portion of the original dataset sourced from PhishStorm and appended to the end of the dataset had phishing and benign labels mistakenly swapped by the author. To validate this, the original PhishStorm dataset was downloaded and manually opened to get a few sample URLs. This was then compared to the Kaggle dataset using command-line tools. This identified that the dataset from PhishStorm was inserted at the end during the making of the Kaggle dataset, and this test verified the labels for PhishStorm data were flipped.

Fixing Dataset Issue:

To fix this issue, an initial attempt was made to correct the wrong labels. However, it was eventually decided to drop the part of the dataset sourced from PhishStorm using command-line tools. This decision is to keep the data intact, lower ambiguity throughout the project, and ensure that model performance is not affected by insufficient data, as training the model on bad data could undermine all model training efforts.

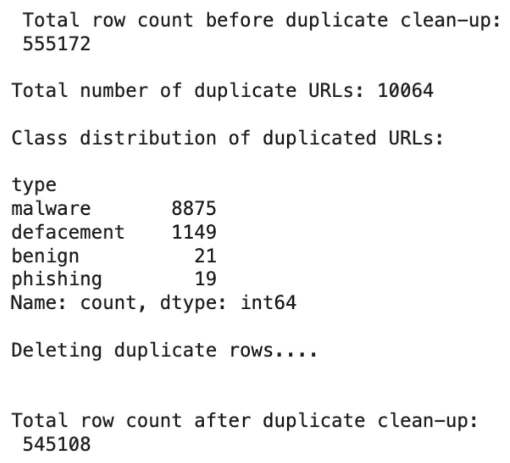
After comparing the two datasets, the last 98,018 rows of the Kaggle set were dropped. The cleaned dataset was uploaded to GitHub to allow import to Google Colab.

Data Cleaning:

The preprocessing steps included deleting rows for duplicate URLs, checking for missing data, and deleting any missing rows.

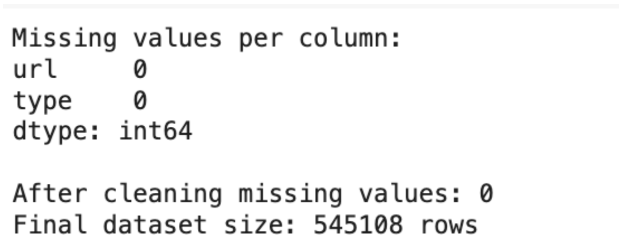
The duplicate URLs check identified that the dataset contained 10,064 duplicated URLs. Out of these duplicates, 8875 were malware URLs. Malware URLs are usually not as common as benign, defacement, or phishing URLs, and this could indicate that malware URLs could be easily identified, use the same tactic, and are reported by several sources, resulting in a higher number of duplicates. These duplicates were dropped to ensure data consistency and prevent data leakage during the split into training and testing sets, and only the first occurrence of these URLs was retained.

The following screenshot shows the output for deleting duplicate URLs.



```
 Total row count before duplicate clean-up:
 555172

Total number of duplicate URLs: 10064

Class distribution of duplicated URLs:

type
malware       8875
defacement    1149
benign          21
phishing        19
Name: count, dtype: int64

Deleting duplicate rows....


Total row count after duplicate clean-up:
 545108
```

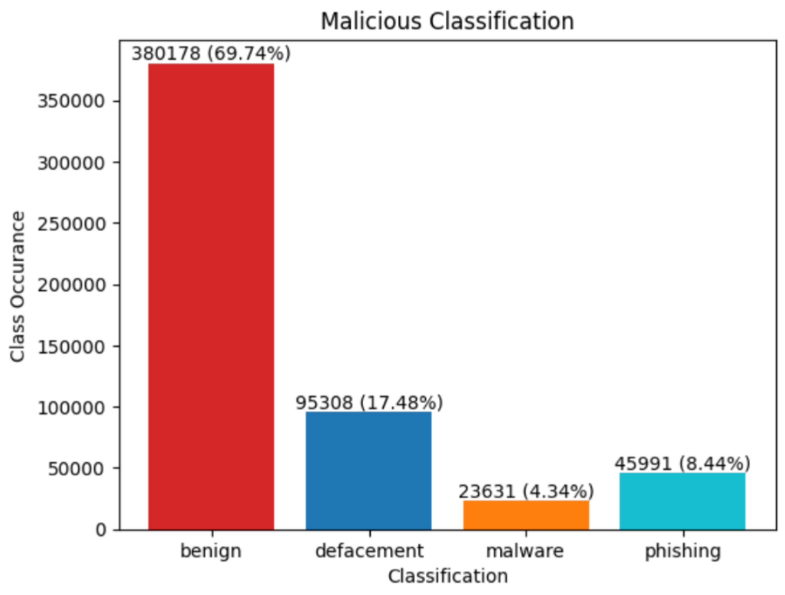
After dropping duplicate URLs, the check for missing data found no missing values in either column. The following screenshot shows the output for missing data.



```
Missing values per column:
url     0
type    0
dtype: int64

After cleaning missing values: 0
Final dataset size: 545108 rows
```

Bar Chart Visualization: Class Balance

After data cleaning was done, the bar chart below was created using the Python Matplotlib library in Jupyter Notebook:



The bar chart shows that benign URLs highest class percentage of 69.74, and malware URLs show the lowest percentage of 4.34. This difference verifies that there is a significant class imbalance in the dataset.

Using the dataset, class imbalance could result in a model biased towards identifying benign URLs, which could impact the detection of malicious URLs.

Balancing Dataset:

To fix the class imbalance issue identified by the bar chart above, 23,000 URLs were randomly sampled from each classification and added to a new DataFrame. The following output shows the total sample size and sample size per classification:

```
Total Sample Size: 92000

Samples per type:
type
benign        23000
defacement    23000
phishing      23000
malware       23000
Name: count, dtype: int64
```

Addressing class imbalance was an essential step in this multi-classification problem to ensure the final model can effectively learn to distinguish between all four categories of URLs. Any processing after this step was done on the balanced dataset.

Dependent Variable Encoding

The dependent variable in this dataset is the column ‘type’, which contains the 4 URL classifications(benign, defacement, malware, phishing). Label encoding was used to convert the classifications into numeric values that the machine learning models can process. The following output shows the label mapping of the classifications:

```
Showing Label Mapping:

benign: 0
defacement: 1
malware: 2
phishing: 3
```

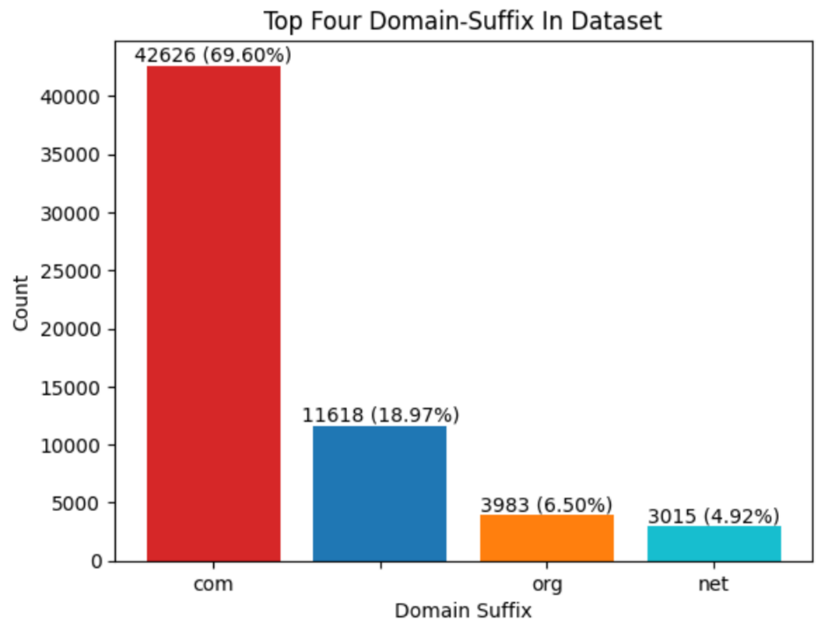
Independent Variable Features Exploration:

The independent variable in the dataset is the column ‘urls’. Since URLs are string data, they needed to be converted into meaningful numeric signals for the model to identify characteristics across the classifications and learn patterns tied to each type from the training set. Manual feature engineering was performed to do this.

The following bar chart was generated to check if domain suffixes could be a good signal to train a model. Domain suffixes are the last part of a website’s URL(e.g., .com, .org).

Using the library, tldextract, the top four domains were identified, and their count was used to plot the bar chart below.

```
Showing the four top domains

 url
com    42626
       11618
org     3983
net     3015
Name: count, dtype: int64
```



The chart indicates that the ‘.com’ domain suffix dominates the dataset with a percentage of 69.6, and 18.97 percent of the URLs show as having missing suffixes. This suggests that most malicious and benign URLs could contain ‘.com’ as the domain suffix, making it more difficult for the model to distinguish based on the domain suffix alone. Thus, the domain suffix alone is not a strong indicator for distinguishing URL types.

Manual Feature Engineering for URLs

Methods were developed to be used to extract URL features. These methods transform raw URLs into meaningful numeric information that the models can use. The extraction methods for this project focused on lexical features(string-based and numeric characteristics).

The following table lists the manual feature engineering methods created, organized by the type of URL characteristic they target.

Category	**Methods**

Length and Character Features	get_url_length()
	count_digits()
	equals_count()
	question_count()
	hyphen_counts()
	count_other_special_char()

HTTPS and Domain Features	https_check()
	count_nums_in_domain()
	domain_is_ip()
	suspicious_suffix()

Keyword Features	count_suspicious()

To target URL domain characteristics, the tldextract library was used, which allowed for accurate separation of subdomain, domain, and suffix elements.

Splitting Data into Train and Test Set:

After defining feature methods, the dataset was split into training and test sets to prepare for model development. 80% of the data was assigned to the training sets, with the remaining 20% used for the test sets. With 92,000 total rows available, 20% is sufficient to test and evaluate the model's performance. random_state from the sklearn Python library was used to apply the same random split each time the program is run(Saturn Cloud, 2024)

The following output shows that the class balance was maintained after the split to train and test:

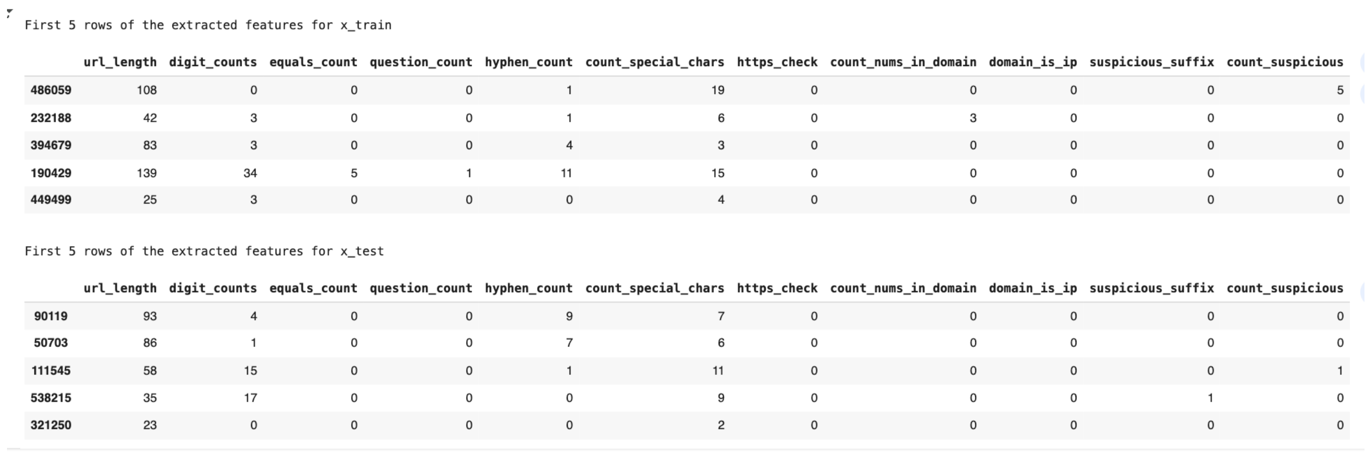
```
Train set class distribution:

Benign        18400
Defacement    18400
Malware       18400
Phishing      18400


Test set class distribution:

Benign        4600
Defacement    4600
Malware       4600
Phishing      4600

Total rows in train set: 73600
Total rows in test set: 18400
```

Maintaining the class balance was important for this project to ensure the model has balanced data to learn and distinguish patterns for each URL classification and avoid biased predictions caused by class imbalance.

Applying Features to train and test sets:

After splitting the dataset, feature extraction methods were applied to both x_train and x_test, and the resulting features were added to new DataFrames.

The following output shows the first five rows of the extracted features displayed from the DataFrame at the time of execution.



First 5 rows of the extracted features for x_train

	url_length	digit_counts	equals_count	question_count	hyphen_count	count_special_chars	https_check	count_nums_in_domain	domain_is_ip	suspicious_suffix	count_suspicious
486059	108	0	0	0	1	19	0	0	0	0	5
232188	42	3	0	0	1	6	0	3	0	0	0
394679	83	3	0	0	4	3	0	0	0	0	0
190429	139	34	5	1	11	15	0	0	0	0	0
449499	25	3	0	0	0	4	0	0	0	0	0

First 5 rows of the extracted features for x_test

	url_length	digit_counts	equals_count	question_count	hyphen_count	count_special_chars	https_check	count_nums_in_domain	domain_is_ip	suspicious_suffix	count_suspicious
90119	93	4	0	0	9	7	0	0	0	0	0
50703	86	1	0	0	7	6	0	0	0	0	0
111545	58	15	0	0	1	11	0	0	0	0	1
538215	35	17	0	0	0	9	0	0	0	1	0
321250	23	0	0	0	0	2	0	0	0	0	0

After features were extracted from the URLs, the data was ready to be applied to a machine learning model.

Machine Learning

Machine learning was applied to the processed data to classify URLs into four categories: benign phishing, defacement, and malware. This was done using a supervised learning process in which the model was trained using labeled data. The model learned from extracted URL features and their associated labels from the training set and used this learning to predict the category of unseen test URLs. The user interface provides a list of sample URLs in a drop-down menu to allow the user to test the functionality, and users can also enter their URLs to get a prediction on their safety. This finished product automates the process of identifying threats from raw URL text.

Using the Python library, scikit-learn, three models were trained and evaluated: Random Forest Classifier, Hist Gradient Boosting Classifier, and Logistic Regression. Training three models allowed for comparison and selection of the most suitable option for this program.

All three models allow for supervised learning from labeled data.

Logistic regression is a linear model used for binary classification. However, scikit-learn allows it to handle multiclass problems by default. Random Forest combines multiple decision trees trained to make a prediction. Histogram Gradient Boosting also uses decision trees and corrects any errors of previous trees during training. The three models were selected because they provide different approaches to classification, ensuring that various algorithms were tested before selecting a final model (Sadananda, 2021).

With the prepared data available, the training for each model involved the following steps:

- Training the model with URL features and encoded classifications from the training sets.
- Applying the model to the URL features in the test set to generate class predictions.
- Comparing the predicted class to the actual class in the test set for evaluation.
- Selecting a final model based on evaluation results.

Random forest was selected as the final model based on its higher performance across different metrics, providing a reliable URL safety prediction.

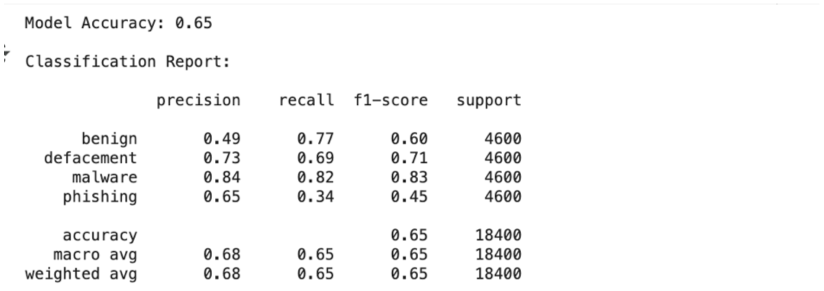
Validation

The three models used in this project(Logistic Regression, Random Forest, and Hist-Gradient Boosting) all fall under supervised learning, as they were trained on labeled data to predict the outcome of new URLs.

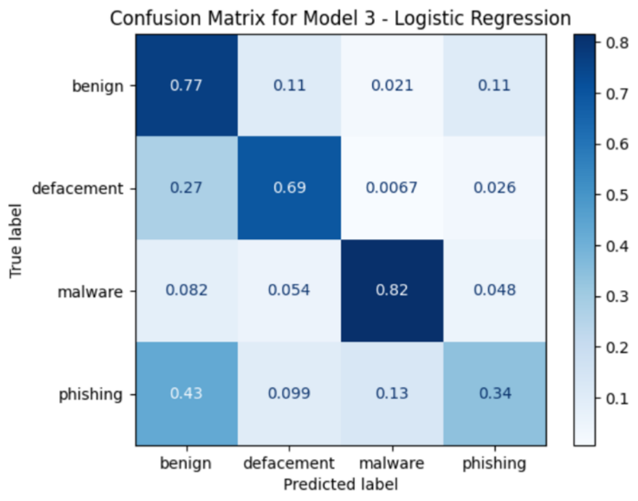
Model performance was evaluated using accuracy, precision, recall, and F-1 score. These metrics reflect the percentage of each model’s correctness and how well each class was identified. A classification report was generated to show each class's precision, recall, and F1-score. A confusion matrix was generated for each mode. A heat map using F1-score was used to help compare the matrices across all models and classes to support final model selection.

Each model's evaluation using a classification report and a confusion matrix, as well as the heatmap, are outlined below:

Logistic Regression Evaluation:

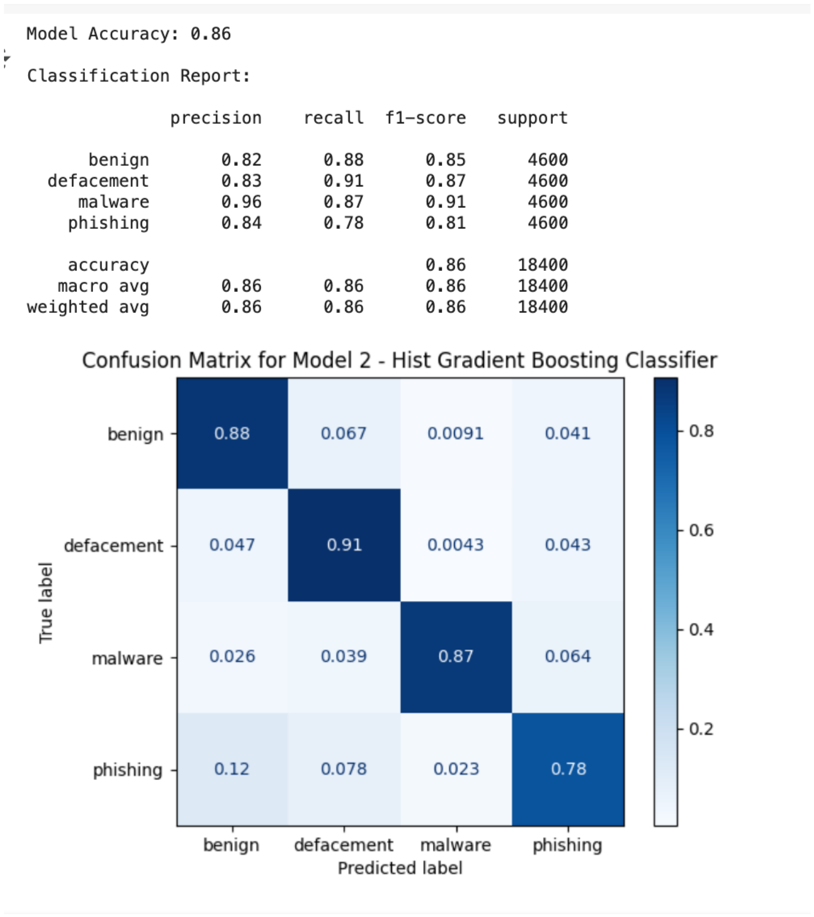


```
Model Accuracy: 0.65

Classification Report:
              precision    recall  f1-score   support

      benign       0.49      0.77      0.60      4600
  defacement       0.73      0.69      0.71      4600
     malware       0.84      0.82      0.83      4600
    phishing       0.65      0.34      0.45      4600

    accuracy                           0.65     18400
   macro avg       0.68      0.65      0.65     18400
weighted avg       0.68      0.65      0.65     18400
```



This model performed the weakest, with an accuracy of 65% and a low F1-score of 0.45 for phishing. This shows that logistic regression may not be the best choice for a dataset with complex features. Since logistic regression is mainly used for linear models with binary classification, it was less effective at learning the patterns in URLs for multiclass classification.

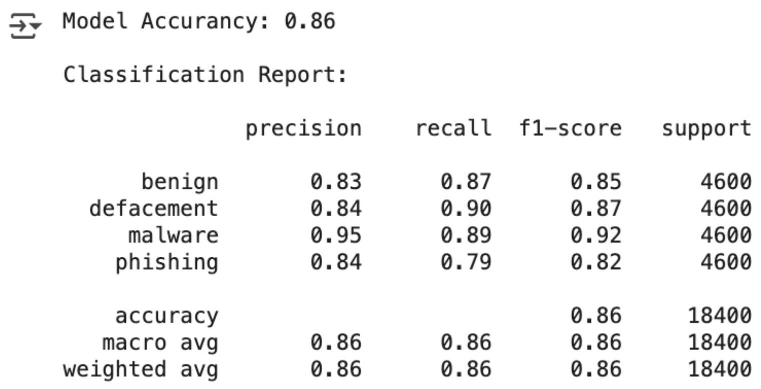
Hist-Gradient Boosting Classifier Evaluation:



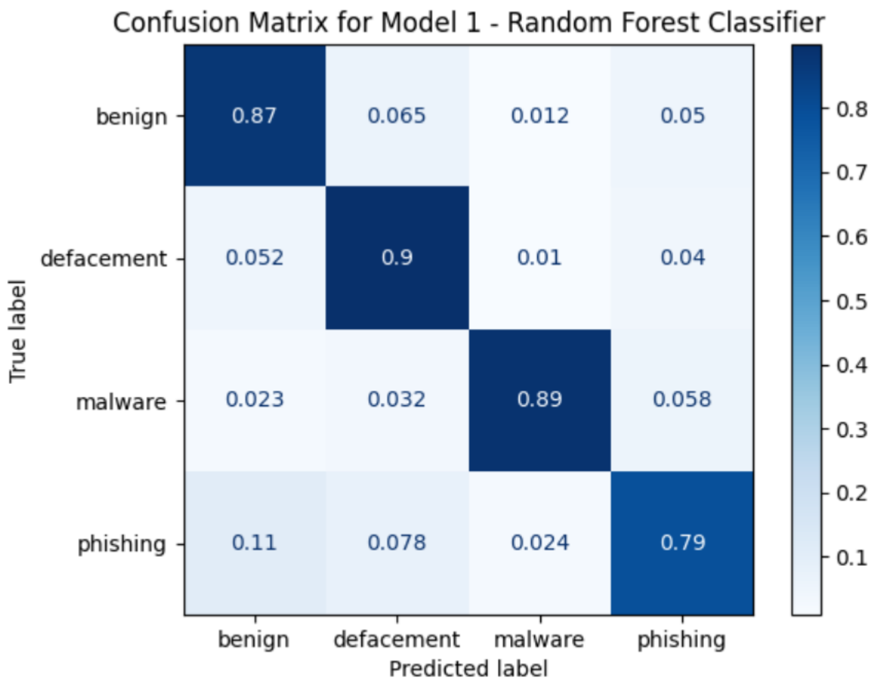
```
Model Accuracy: 0.86

Classification Report:

              precision    recall  f1-score   support

      benign       0.82      0.88      0.85      4600
  defacement       0.83      0.91      0.87      4600
     malware       0.96      0.87      0.91      4600
    phishing       0.84      0.78      0.81      4600

    accuracy                           0.86     18400
   macro avg       0.86      0.86      0.86     18400
weighted avg       0.86      0.86      0.86     18400
```

Hist Gradient Boosting achieved an accuracy of 86%. Malware showed the highest F1-score(0.91), indicating strong detection ability. Phishing had the lowest F1-score(0.81) among the four classes, but it shows a big improvement over the logistic regression model. This suggests that the model, at times, may struggle to identify phishing URLs, which is expected, as most phishing URLs present themselves as safe and rely on social engineering. Benign and defacement classes showed stable performance.

Random Forest Classifier Evaluation Report:

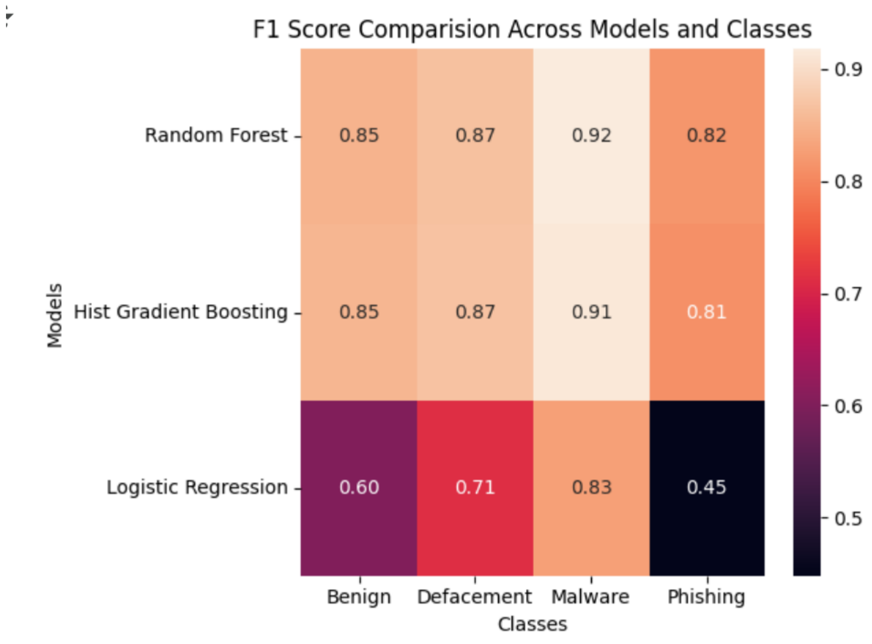


```
Model Accurancy: 0.86

Classification Report:

              precision    recall  f1-score   support

      benign       0.83      0.87      0.85      4600
 defacement       0.84      0.90      0.87      4600
     malware       0.95      0.89      0.92      4600
    phishing       0.84      0.79      0.82      4600

    accuracy                           0.86     18400
   macro avg       0.86      0.86      0.86     18400
weighted avg       0.86      0.86      0.86     18400
```



The model achieved an accuracy of 86%. Malware here also shows the highest F1-score(0.91), with benign and defacement classes showing good performance. The consistent high performance of malware detection shows that the models can easily identify malware URLs. This could also indicate that malware URLs have distinct lexical features that make it easy to identify them. Phishing again had the lowest F1-score(0.82), which was influenced by its low recall.

Heat Map for Model Comparison:



The heatmap compares the F1-score across all models and classes. It highlights the weaker performance of Logistic Regression and shows that the tree-based models(Random Forest and Hist Gradient) performed consistently across all categories.

Chosen Model for Deployment:

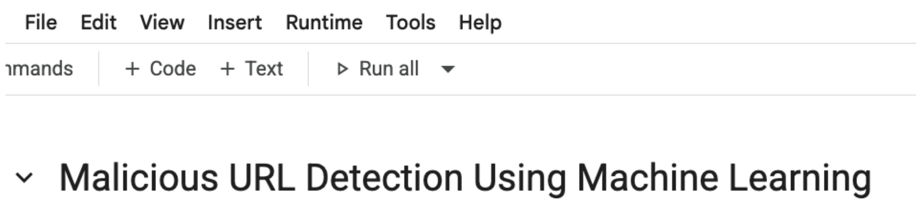
Based on the evaluation reports, I selected the **Random Forest Classifier** for the URL detection tool. While Random Forest and Hist-Gradient Boosting performed similarly, Random Forest shows a slightly increased score across classes. Logistic Regression performed significantly worse overall.

User Guide

The final application is a Jupyter notebook hosted on Google Colab. Once the notebook is run, it presents the user with an input field to either enter a URL or select from the sample list provided in the drop-down menu.

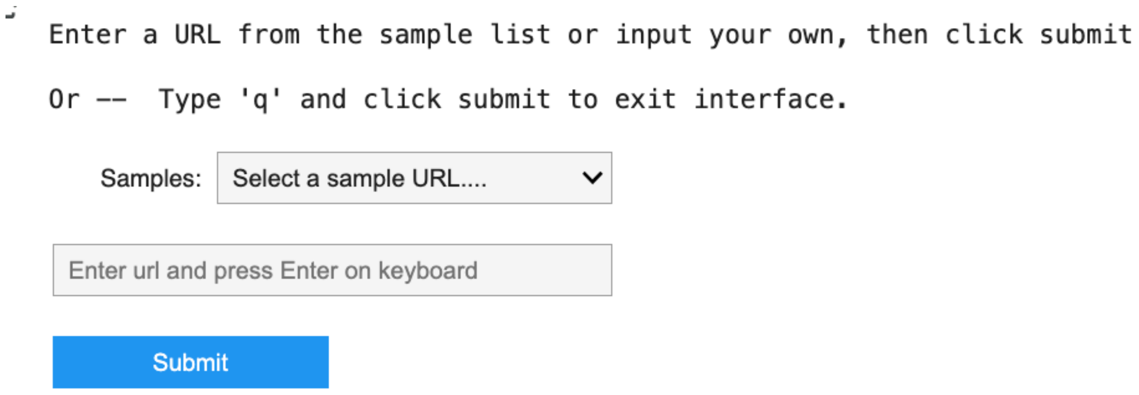
User's Guide:

1. Use the following link to access the Google Colab: https://colab.research.google.com/drive/1a4t6rzFe1__GxFr8nJK5yuCtVprbDquw?usp=sharing
2. Once you're on the dashboard, click 'Run all'.



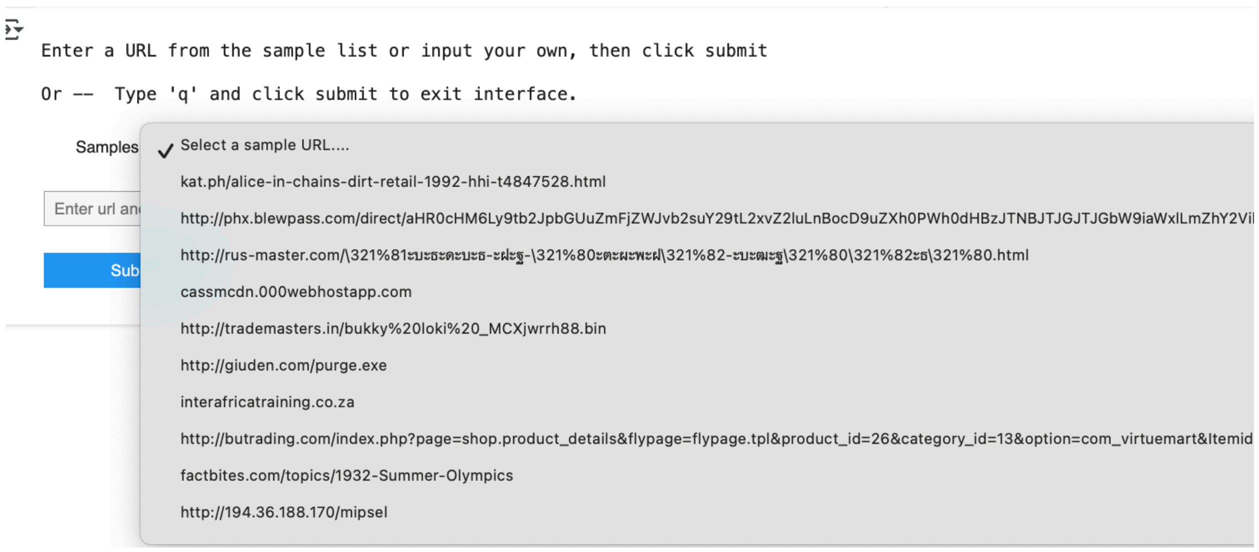
Malicious URL Detection Using Machine Learning

3. You may see a security warning that says, 'This notebook was not authored by Google..' and options to Cancel or Run anyway.
4. Click 'Run anyway' – This will start by installing and importing libraries and running all the cells.
5. Once the run is completed, the last cell will prompt the user to enter a URL



```
Enter a URL from the sample list or input your own, then click submit

Or --  Type 'q' and click submit to exit interface.
```

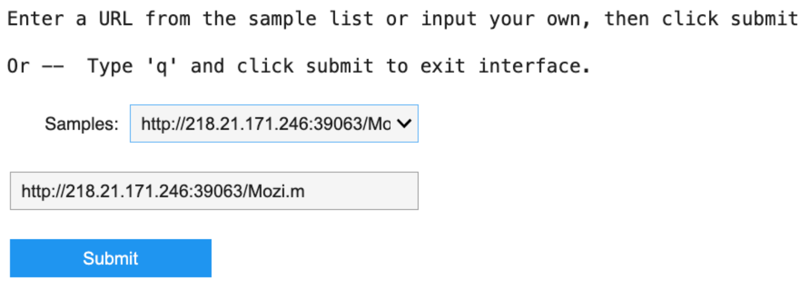
6. You may select a URL from the samples provided in the drop-down menu shown below or enter your new URL. The sample list may differ at the time of run since resampling is done with each rerun.



```
Enter a URL from the sample list or input your own, then click submit

Or --  Type 'q' and click submit to exit interface.
```

Samples
Enter url an
Sub
✓ Select a sample URL....
kat.ph/alice-in-chains-dirt-retail-1992-hhi-t4847528.html
http://phx.blewpass.com/direct/aHR0cHM6Ly9tb2JpbGUuZmFjZWJvb2suY29tL2xvZ2luLnBocD9uZXh0PWh0dHBzJTNBJTJGJTJGbW9iaWxlLmZhY2Vi
http://rus-master.com/\321%81ะบะธะดะบะธ-ะฝะฐ-\321%80ะตะผะพะฝ\321%82-ะบะฒะฐ\321%80\321%82ะธ\321%80.html
cassmcdn.000webhostapp.com
http://trademasters.in/bukky%20loki%20_MCXjwrrh88.bin
http://giuden.com/purge.exe
interafricatraining.co.za
http://butrading.com/index.php?page=shop.product_details&flypage=flypage.tpl&product_id=26&category_id=13&option=com_virtuemart&Itemid
factbites.com/topics/1932-Summer-Olympics
http://194.36.188.170/mipsel

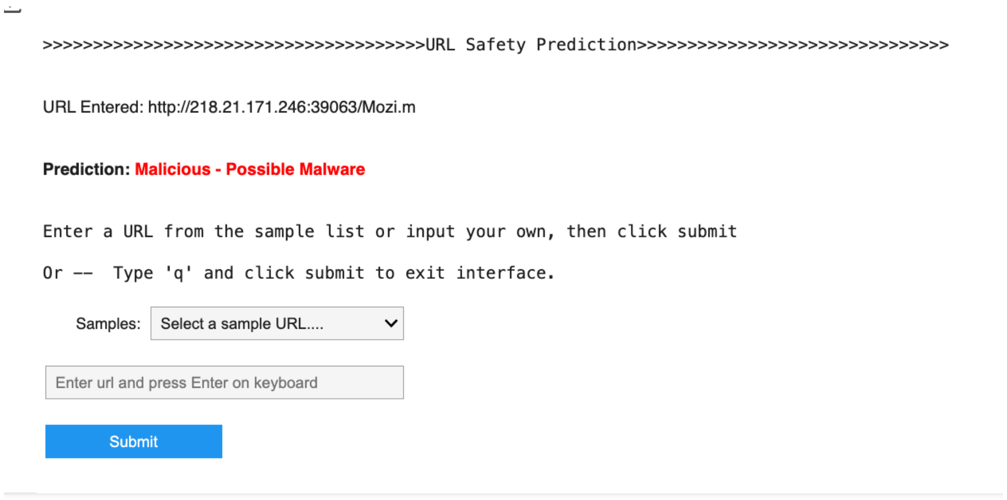
7. Once the URL is entered, click the ‘Submit’ button



```
Enter a URL from the sample list or input your own, then click submit

Or --  Type 'q' and click submit to exit interface.
```

Samples:
http://218.21.171.246:39063/Mo
http://218.21.171.246:39063/Mozi.m
Submit

8. The safety prediction is shown as follows:



```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>URL Safety Prediction>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

URL Entered: http://218.21.171.246:39063/Mozi.m

Prediction: Malicious - Possible Malware

```
Enter a URL from the sample list or input your own, then click submit
Or --  Type 'q' and click submit to exit interface.
```

9. You may check more URLs or type q, and press Enter to exit the URL checker.

Reference Page

Sadananda, N. (2021, September 29). *The battle between logistic regression, random forest classifier, xg boost and support vector...* Medium. https://medium.com/@nischitasadananda/the-battle-between-logistic-regression-random-forest-classifier-xg-boost-and-support-vector-46d773c70f41

Saturn Cloud.(2024, January 9). *What is “random_state” in sklearn.model_selection.train_test_split example?* Saturn Cloud Blog. https://saturncloud.io/blog/what-is-randomstate-in-sklearnmodelselectiontraintestsplit-example/