Catch it!

Malicious URL Classifier

Developer: Kejin Qian

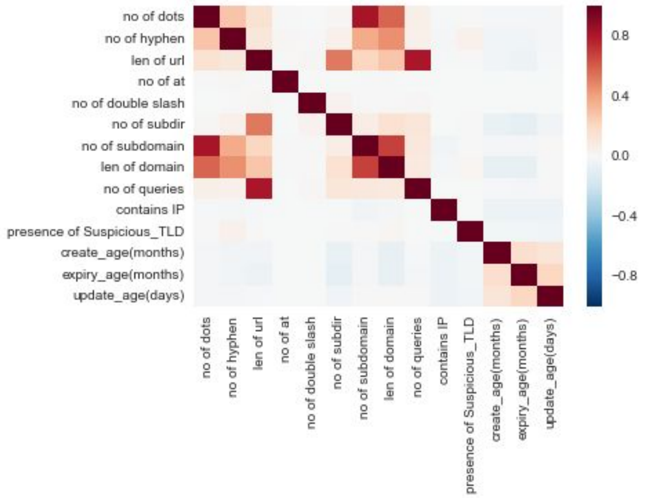
1

Highlights

1

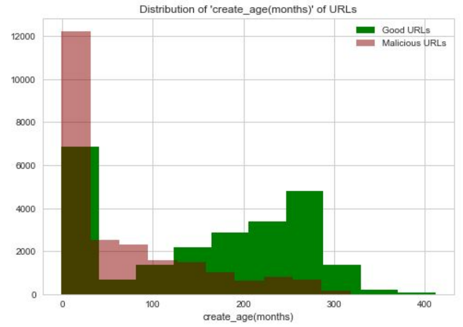
- Explored on special characteristics that malicious URLs usually have.
- Extracted 14 features from URL strings and obtained Host based features by querying Whois Server.

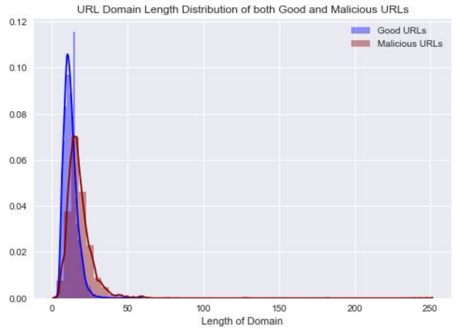
1. IP address used as an alternative of the domain name
2. Existence of @, //, - in domain names
3. Large number of query strings in URL



2

- Created visualizations on distributions of features obtained of both malicious and good urls.
- Enabled users to see the significance of each feature based on the aggregated url database.
- Deterministic features of malicious URL:





3

- A set of machine learning predictive models was built, a 10-fold cross validation was applied to each model to evaluate model performance.
- Model Summaries:

Model	Accuracy	AUC
Decision Tree	90.08%	0.918
Random Forest	92.55%	0.980
Adaboost	89.85%	0.965
Gradient Boosting	90.64%	0.970
Logistic Regression	84.70%	0.928

2

Review Process

Completed Work

- **Data Acquisition**
  - Aggregated lists of verified malicious and safe urls from the following sources.

    Phishtank https://www.phishtank.com/ the Majestic Million https://majestic.com/reports/majestic-million?s=1000 and https://github.com/incertum/cyber-matrix-ai
  - Final dataset summary: 47273 URLs in total, 49.6% of urls are malicious urls.
- **Feature Generation based on URL string**
  - Extracted 11 features from the URL string and created 3 Host-based features by making queries from WHOIS server
  - Create visualizations on distributions of each feature obtained of both malicious and secure urls. Enable users to see the significance of each feature on classifying malicious urls based on the aggregated url database.
  - Aggregated lists of verified malicious and safe urls from the following sources.
- **Exploratory Data Analysis**
  - Ran descriptive analyses across all collected features and implemented missing-data imputation.
  - Completed feature selection, standardization and feature transformation(log) to correct the skewness.
- **Predictive Modeling**
  - Trained and tuned (via Cross Validation) each potential candidates of classification algorithms(Decision Trees, Random Forest, AdaBoost, Gradient Boosting and Logistic Regression).
  - Completed model selection using performance metrics AUC and classification accuracy. The optimal model is Random Forest.

3

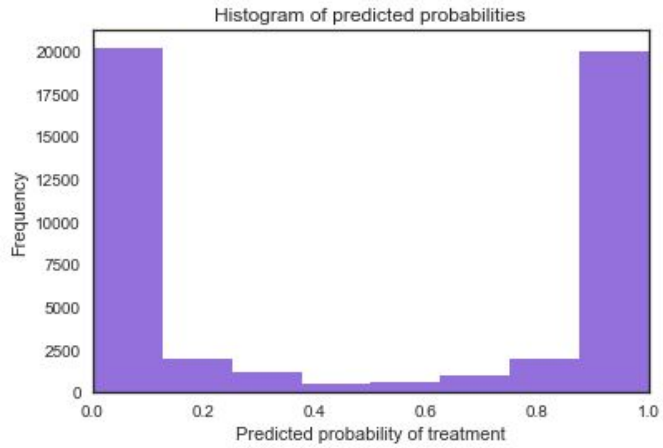
Demo/analysis

Best Model Performance Summary

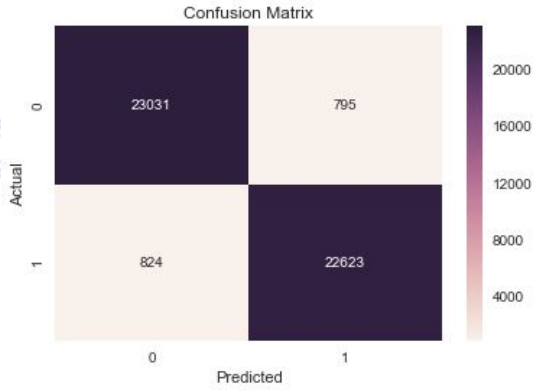
```
+--------------------+--------------------+
|    Random Forest   | n_estimators = 150 |
+--------------------+--------------------+
| 10-fold CV Accuracy|       92.795%      |
|    False Positive  |        2.296%      |
|    False Negative  |        2.286%      |
|       F1 Score     |        0.977       |
|         AUC        |        0.977       |
+--------------------+--------------------+
```

We want the predicted probabilities to be close to 0 or 1, so that our prediction is made with confidence.

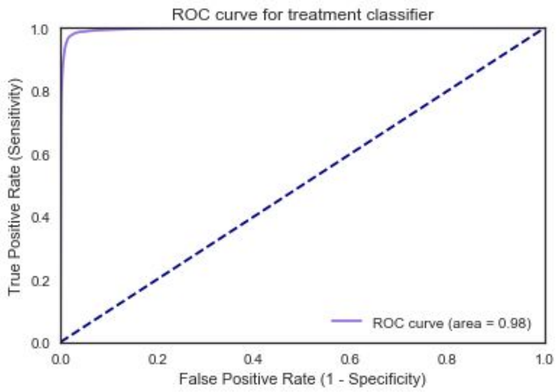
Histogram of Predicted Probabilities



Confusion Matrix

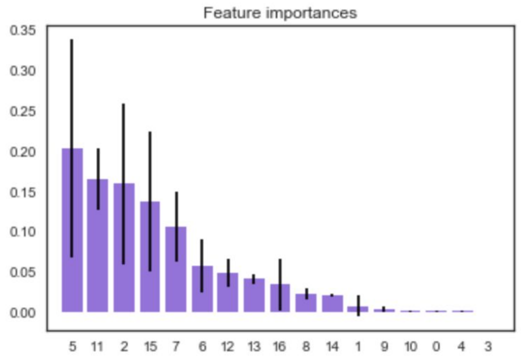


ROC Curve



Feature Importance

```
Feature ranking:
1. feature no of subdir (0.202283)
2. feature create_age(months) (0.164614)
3. feature len of url (0.158797)
4. feature file extension (0.136176)
5. feature len of domain (0.105130)
6. feature no of subdomain (0.057108)
7. feature expiry_age(months) (0.048354)
8. feature update_age(days) (0.040318)
9. feature risk indicator (0.033404)
10. feature no of queries (0.022276)
11. feature country (0.020440)
12. feature no of hyphen (0.006946)
13. feature contains IP (0.003041)
14. feature presence of Suspicious_TLD (0.000415)
15. feature no of dots (0.000350)
16. feature no of double slash (0.000344)
17. feature no of at (0.000005)
```



4

Lessons Learned

- **About the Malicious URL Classifier**
  - Based on the model performance, we could say that malicious URL detection can be done with great confidence even if we don't have any information about the content of the webpage. This model helps to optimize the detection process since downloading or crawling the web content is time-consuming.
  - Specifically to this project, visualization of features are extremely helpful in feature selection and modeling processes. These visualizations are also convincing evidence to why some features were given high variable importances in the Random Forest model.
- **About the Project Management**
  - Experienced the benefits of having reproducible datasets by writing python scripts to get source data via URL.
  - In the early development stage, using Jupyter Notebook brings lots of effectiveness and efficiency. But I learnt how to convert them into several functions and write them in .py scripts. Running .py scripts in terminal is very effective.
  - Learned how to use effective documentations through docstring, comments and git commit messages to optimize the quality and understandability of my codes. These documentations are extremely helpful in the QA process.
- **About the Technology**
  - Better understanding on how EC2, S3 and RDS work individually and how they can be connected to achieve better project management.
  - Learned how to create database schema and insert data into the database locally and in RDS.

5

Recommendations

Next Step

- **Project Side**
  - Finalize the classification model and learn to make it as a reproducible model after doing Assignment 3.
  - Querying from WHOIS server to acquire host-based features is very time-consuming(Spent 9 hours to get all the features for ~40k URLs). Need to think about how to optimize the app response time (prediction time) especially when user input multiple URLs at the same time.
  - Use RDS to backup my model, and create the URL database in RDS which initially takes all my training samples, then updates every time with new user inputs and corresponding prediction results.
  - Build the app of my Malicious URL Classifier using Flask and HTML, which takes single or multiple URL(s) as input and outputs the classification results and explanations.

- **Code/Script Side**
  - Start to rewrite everything in jupyter notebooks to functions, categorize them and store them in multiple python scripts. Using helper functions to avoid redundancy in codes and maximize the effectiveness and efficiency.
  - Add unit tests with edge cases for the model and other python scripts. These tests should be run and get passed locally and also in a newly created environment(a full requirements.txt file needs to be prepared in the next step).