An Automated Framework for Real-time Phishing URL Detection

Farhan Sadique*, Raghav Kaul†, Shahriar Badsha‡, Shamik Sengupta§
Dept. of Computer Science and Engineering
University of Nevada, Reno, NV, USA
fsadique@nevada.unr.edu*, raghavkaul@nevada.unr.edu†, sbadsha@unr.edu‡, ssengupta@unr.edu§

***Abstract*—An increasing number of services, including banking and social networking, are being integrated with world wide web in recent years. The crux of this increasing dependence on the internet is the rise of different kinds of cyberattacks on unsuspecting users. One such attack is phishing, which aims at stealing user information via deceptive websites. The primary defense against phishing consists of maintaining a black list of the phishing URLs. However, a black list approach is reactive and cannot defend against new phishing websites. For this reason, a number of research have been done on using machine learning techniques to detect previously unseen phishing URLs. While they show promising results, any such implementation is yet to be seen. This is because 1) little work has been done on developing a complete end-to-end framework for phishing URL detection 2) it is prohibitively slow to detect phishing URLs using machine learning algorithms. In this work we address these two issues by formulating a robust framework for fast and automated detection of phishing URLs. We have validated our framework with a real dataset achieving 87% accuracy in a real-time setup.**

***Index Terms*—phishing, malicious, url, online learning, machine learning, cybersecurity**

I. Introduction

World wide web plays a major role in modern life. It offers numerous functionalities including social networking, banking, and e-commerce. The primary reason for widespread adoption of internet is the flexibility it offers. For instance, we are presently able to perform major financial operations from the convenience of our homes. However, such flexibility comes at a price in the form of cyberattacks. A large number of unsuspecting users fall victim to various cyberattacks every year. McAfee and the Center for Strategic and International Studies (CSIS) calculated the global annual cost of cybercrimes to be almost 600 billion US dollars [1].

A major type of cyberattack, that affects the people and businesses alike, is phishing [2]. The attacker of a phishing attack tries to gather sensitive information of an user by disguising as a trustworthy third party. Such an attack usually consists of directing users to a fake website that resembles another legitimate website. The URL of the fake, phishing website is typically distributed via emails or instant messages. Oftentimes it is very difficult or impossible for a user to detect such fake websites just by looking at the content of the web page. Phishing attacks cost a mid-sized company about 1.5 million US dollars on average [3].

This research is supported by the National Science Foundation (NSF), USA, Award #1739032.

It should be noted here that phishing URLs are part of a broader set called malicious URLs. Other types of malicious URLs include drive by download URLs, spam URLs etc. Even though the rest of the work discusses detecting phishing URLs only, the generalized procedure can easily be tweaked for detecting all kinds of malicious URLs.

The most popular technique used to detect phishing URLs is the use of blacklists [4]. A blacklist is simply a list of malicious URLs, periodically updated by community users or cybersecurity experts. However, the Webroot Threat Report estimates [5] that nearly 1.5 million phishing websites are created every month. As a result it is not possible to blacklist all phishing websites and corresponding URLs in a timely manner. So, an automated framework to detect new phishing URLs is required. Such a system should detect previously unseen phishing URLs with high accuracy without any human interaction.

Many approaches have been explored towards such an end including machine learning [6]–[10]. A machine learning approach begins with collection of a dataset. The dataset includes different features and labels (benign vs. malicious) of a large number of URLs. A classifier is then trained with the collected dataset.

The underlying assumption is that a malicious URL has a significantly different feature distribution than a benign URL. As a result a good machine learning approach should be able to differentiate between benign and malicious URLs based on those features. This hypothesis has been verified in numerous previous works [4], [11]–[15] using diverse dataset and state-of-the-art machine learning algorithms.

A. Challenges & Motivation

While many of the previous works show promising results, adoption of such a mechanism in the industry is yet to be seen. This is due to the fact that -

1) A complete, standard framework for detecting malicious URLs has not been proposed.
2) It is prohibitively slow to detect malicious URLs using machine learning algorithms in a real-time setup.
3) The URL space is highly unbalanced with many more benign URLs than phishing URLs.
4) The URL space is dynamic and changes over time, meaning the classifier must be updated periodically.

978-1-7281-3783-4/20/$31.00 ©2020 IEEE

5) The growth of the URL space is unbounded, which means we cannot use traditional batch learning methods to train on all URLs.

This work aims to tackle these challenges by introducing a complete automated framework for phishing URL detection. While the framework is built for detecting phishing URLs only, the general approach applies to identifying all types of malicious URLs.

B. Contribution

The primary contribution of this work is introducing a complete, automated, real-time framework to detect phishing URLs. Other major contributions of this work are:

1) We formulate the design of a framework to automatically detect phishing URLs.
2) We use an online or incremental learning method to tackle the unbounded growth of the URL space.
3) We propose a delayed feature collection algorithm to increase performance of our system.
4) We propose a selective sampling algorithm to further improve the performance.
5) We test our system's accuracy and performance with a very diverse and unbiased dataset.
6) We analyze the feature importance to gain meaningful insights into the dataset.

Our system is able to label an URL with 87% accuracy without suffering from any performance bottleneck.

This work outlines the general approach and leaves room for improvement to future researchers in this field. Since, our work clearly outlines the compromise between accuracy and performance of the detection process, it is possible to tune this system to achieve acceptable accuracy without conceding considerable speed or performance.

II. Related Work

Many previous researchers have studied the detection of phishing URLs and malicious URLs in general. Many of these works used machine learning approaches to detect malicious URLs. The performance of their work depends primarily on the feature set, the dataset and the particular algorithm used.

Ma et al. [12] compared three classifiers, namely Naive Bayes (NB), support vector machine (SVM) and logistic regression (LR) on a very good dataset. They used features like bag of words (BOW), IP address, WHOIS information, domain characteristics and geolocation. They cross verified the results using two datasets created from 4 sources. However, the simple classifiers used in their work is not suitable for deployment. Furthermore, the dataset had a ratio of $1:3$ for malicious and benign URLs. In reality any such a system will see many more benign URLs than malicious URLs.

Tan et al. [15] further tested many more classifiers including AdaBoost (ADB), decision tree (DT), gradient boosted trees (GDB), perceptrons (PE), K-nearest neighbor (KNN), and random forest (RF). ADB performed best in their scenario. However, the drawback of their work is they worked with a very large dataset with very limited number of features (24). Such a training often suffers from over fitting. In other words their algorithm is comparable to a traditional blacklist because of the massive dataset they used.

Sahingoz et al. [16] were one of the first to use natural language processing (NLP) on URLs to collect features. They also adopted a heuristic approach to detect common brand names in URLs. Another novel contribution of their work is detecting typosquatting. Typosquatting is a technique that malicious websites use to catch typos in legitimate URLs. They also collected a large dataset by querying a search engine. Hence, it is susceptible to bias of user query behavior. Furthermore, their dataset was balanced as some other works. Finally, they only used 10% of the collected data because of performance limitation.

Xiang et al. [13] introduced two new features in detecting malicious URLs – 1) HTML content of the actual page and 2) PageRank [17] of the URL. Pagerank has proven to be a very promising feature in detecting malicious URLs as they usually have a very low PageRank. The novel contribution of their work was detecting duplicate malicious websites really fast from previously created hashes of visited websites. As a result their algorithm can in theory detect the reused malicious content in different URLs. However, their dataset was very small and the heuristic algorithm to detect duplicate websites was very specific.

Mamun et al. [18] used only lexical features derived from the URL itself for fast detection of malicious URLs. They hypothesized that the URL itself contains enough features to differentiate between a benign and a malicious URL. However, their benign URL dataset was highly biased with pages from a handful high ranked websites. Moreover, a recent report by WebRoot revealed that nearly 40 percent of all malicious URLs are found on compromised good domains [19].

Zhao et al. [20] used time-varying features of a URL to detect if it becomes malicious over-time. They also came up with a good training metric to tackle the unbalanced nature of the dataset with many more benign URLs than malicious URLs. They were also the first to use an online learning algorithm with selective sampling for fast detection. While the selective sampling of malicious URLs was a significant contribution they used an existing dataset with little to no modification of the algorithm for URL detection.

Jeeva et al. [21] also used only lexical features derived solely from the URL to differentiate between malicious and benign URLs. However, their approach was statistical in nature and they came up with 14 statistics of the dataset to differentiate between malicious URLs manually. Although really fast, this approach is prone to human error and will fail to adapt to the ever changing nature of the world wide web. Any ensemble technique like random forest should outperform their algorithm in the long run.

III. System Architecture

The functional diagram in figure 1 shows the system architecture of our proposed system. The architecture is explained in the following subsections:

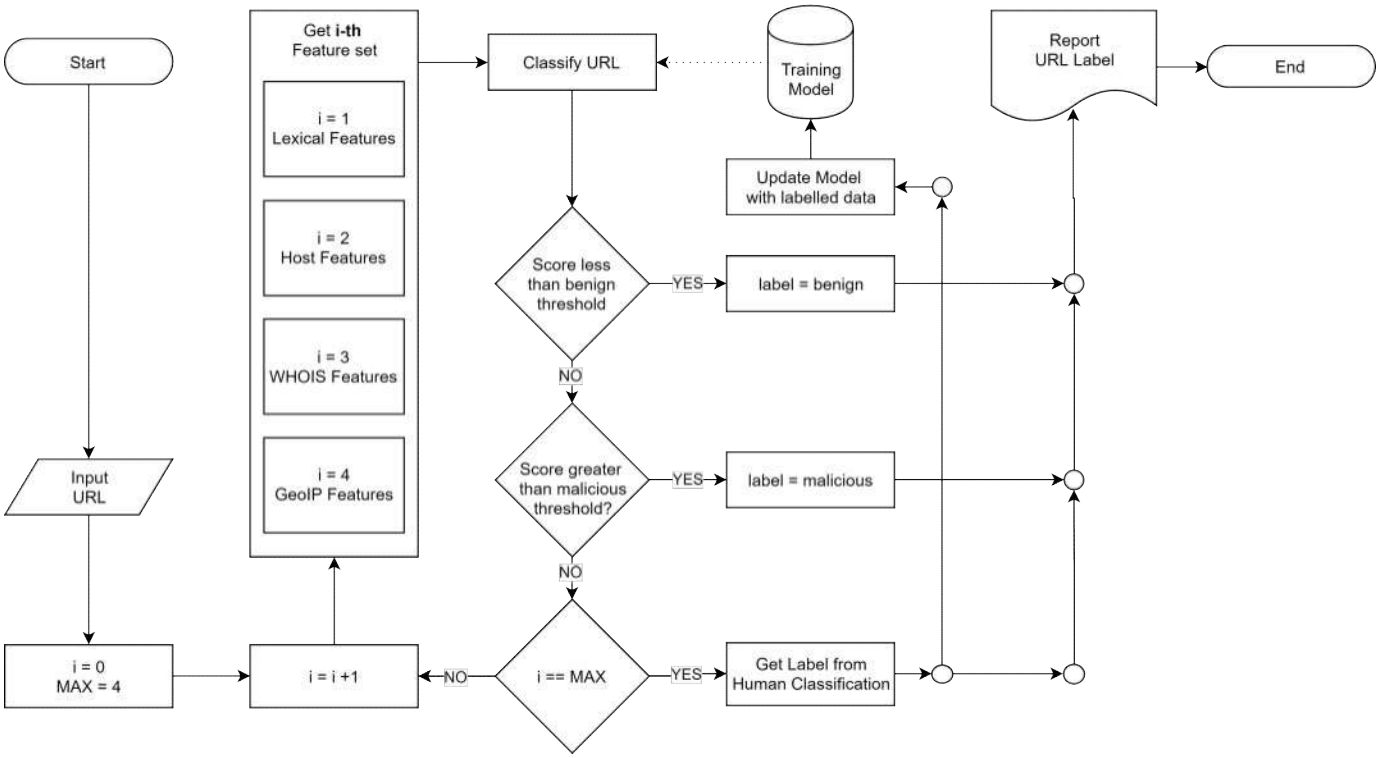

Fig. 1. Functional Diagram of System Architecture

A. Overview of Data Flow

The process begins with an input URL. The URL can be input by an user via an API that we built. Then, we collect the 1^{st} set of features ($i = 1$ in figure 1). The description of the feature sets and how they are ordered is given in III-B. We then try to classify the URL with this 1^{st} set of features only. Rather than getting a binary classification $label \in 0, 1$ we calculate its $score \in \mathbb{R}$ and $score \in [0, 1]$. The $score$ denotes the probability of the URL being malicious.

It should be noted here we use an online/incremental learning classifier instead of a batch learning classifier in our system. This is because a batch learning classifier will become obsolete, as the feature space of URLs change over time. However, since our dataset grows in an unbounded manner, it is impractical to retrain a batch learning classifier periodically. On the other hand an online learning classifier can be updated with one data sample, making it suitable for the task at hand.

This $score$ is then used to decide if the URL can be classified as either benign or phishing with high confidence. If the confidence is low, the next set of features are collected and the process continues till there are no more feature sets to collect. Although, we worked with 4 sets of features, we can easily extend the framework to include an arbitrary number of feature sets.

If we fail to classify the URL with high confidence after collecting all features, we request the $label$ from a human. The human-$label$ can be requested from a system administrator or decided from user votes. The $label$ is considered ground truth is used to update our classifier training model. If our classifier can classify the URL with high confidence at any stage, it stores the $label$ and reports it to the user.

Performance Consideration: We do not collect all the features right away because more time is required to collect the latter feature sets than the former ones. So, we try not to collect the latter features right away. This is further explained in III-D

We also do not update our classifier model with all the URLs. We only do so when our algorithm fails to classify it completely. In that case we request to $label$ to a human. We than update our classifier with that URL's features and $label$. This selective sampling technique is explained in III-E.

B. Feature Sets

As seen in figure 1, we work with 4 sets of URL features: lexical, host, GeoIP, and domain WHOIS. Each feature set is described below:

1) Lexical Features: Lexical features are based on the URL string itself. Several examples of typical lexical features are number of characters in the URL, number of dots in the URL and number of symbols in the URL. They can be related to not only the whole URL but specific parts of it also like the domain, the filename or the query. We collect several such features from the URL string itself.
2) Host Based Features: Host based features are based on the server that hosts the webpage pointed by the URL. The simplest such feature is the IP address that the URL resolves to.

3) Domain WHOIS Based Features: Any Regional Internet Registry (RIR) like ARIN or APNIC maintains it's own domain WHOIS database that contains information on the domain registrant. In addition to general information like name of registrant, the database also includes the creation and expiration dates of the domain registration. These data can be queried using the RDAP [22] protocol. We analyze these data to parse a great number of features. Examples are 'number of days before the domain registration expires' and 'number of days passed after creation of the domain'.
4) GeoIP Based Features: GeoIP features are obtained from the IP address of the host. We used the GeoLite2 [23] IP geolocation databases from MAXMIND to collect various GeoIP features of the host IP address. GeoIP features include autonomous system number (ASN), country, city, latitutde, longitude etc.

These feature sets are used in our system to classify the URLs with reasonable accuracy. Since, our primary goal is to propose and validate a framework, we choose to do so with an adequate number of features. However, our process being linear, it is possible to add more feature sets in future.

C. Online Machine Learning Classifier

Phishing URL detection is a dynamic problem. This means we need to update our classifier model periodically. However, since the URL space grows without bound we cannot train a batch learning classifier indefinitely. In contrast, an online/incremental learning [24] classifier can be updated iteratively.

If, $\mathbf{wt_{n-1}}$ is a prediction model trained with $n-1$ instances of data, an online learning algorithm can update it for the n^{th} data instance with features $\mathbf{xt_n}$ and label yt_n as follows:
$\mathbf{wt_n} = \mathbf{wt_{n-1}} + f_o(\mathbf{xt_n},\ yt_n)$, where f_o is some update function. On the other hand a batch learning algorithm achieves the same as follows:
$\mathbf{wt_n} = f_b(\{\mathbf{xt_1},\ \mathbf{xt_2},\ ...\ \mathbf{xt_n}\}, \{yt_1,\ yt_2,\ ...\ yt_n\})$, which is computationally infeasible because n grows without bound.

D. Delayed Feature Collection

Our system achieves improved performance during real-time URL classification, because of how we collect the feature sets. Our algorithm assigns a *cost* of collection to each feature set. We calculate *cost* as the time required to collect the corresponding set of features. We ignored the required processor cycles from *cost* calculation because the feature collection process is I/O limited, not processor limited.

On average our system collects the 'lexical features' of an URL in 3 ms. In contrast it takes about 1453 ms to collect the 'WHOIS features' of an URL on average. Thus, the *cost* estimation and corresponding ordering of feature sets have significant impact on the real-time performance of our system.

The relative ordering of the feature sets in figure 1 is based on their respective *cost*. As shown in the figure, our algorithm tries not to collect the more expensive feature sets unless necessary. We try to classify the URLs with the less expensive feature sets first before collecting the more expensive ones. This improves the performance and responsiveness of our system.

Algorithm 1 SELELCTIVE SAMPLING

▷ Classifier model for online learning
1: global **clf**
▷ Get i^{th} feature set of url u
2: **procedure** GETFEATURES(u, i)
▷ Update classifier with features $\mathbf{x}$ of url u
3: **procedure** FITONE(**clf**, $\mathbf{x}$, $label$)
▷ Probability that url u with features $\mathbf{x}$ is malicious
4: **procedure** PREDICTPROBA(**clf**, $\mathbf{x}$)
▷ Get $label$ of url u from human classification
5: **procedure** REQUESTLABEL(u)

6: u ▷ URL data sample in consideration
7: $MAX \leftarrow 4$ ▷ Number of feature sets
8: $\epsilon_b[MAX]$ ▷ u is benign if $score < \epsilon_b[i]$
9: $\epsilon_m[MAX]$ ▷ u is malicious if $score > \epsilon_m[i]$

▷ Initialize feature vector $\mathbf{x}$ of length n to all -1 (NaN)
10: $i \leftarrow 0$
11: init float array $\mathbf{x}[n]$
12: **for** $j \leftarrow 1$ to n **do**
13: $\mathbf{x}[j] \leftarrow -1$

14: **do**
15: $i \leftarrow i + 1$
16: $\mathbf{xi}, n_i \leftarrow$ GETFEATURES(u, i)
17: **for** $j \leftarrow 1$ to n_i **do**
18: $\mathbf{x}[j] \leftarrow \mathbf{xi}[j]$
19: $score \leftarrow$ PREDICTPROBA(**clf**, $\mathbf{x}$)
20: **while** $(\epsilon_b[i] \leq score \leq \epsilon_m[i]$ **or** $i \leq MAX)$

21: **if** $i > MAX$ **then** ▷ Automatic classification failed
22: $label$ = REQUESTLABEL(u)
23: FITONE(**clf**, $\mathbf{x}$, $label$)

E. Selective Sampling

We further employ selective sampling to improve the performance of our real-time system. The concept was first explored by Zhao et al. in [20]. However, we have developed a simpler algorithm that generalizes to any online/incremental learning classifier. The algorithm 1 shows the pseudocode for how we did selective sampling.

The parameter sets malicious threshold, ϵ_m and benign threshold ϵ_b lie in the heart of the selective sampling algorithm. We need to maintain these thresholds after getting each feature set (for each $i \in 1, 2, 3, 4$ in our case).

For a particular i and for a particular $label$ (benign or malicious), we can incrementally maintain the mean and the variance of the probability scores using following formulae:

$$\overline{x}_n = n^{-1}(s_n + (n-1) \times \overline{x}_{n-1}) \quad (1)$$

$$s_n^2 = \frac{n-2}{n-1} s_{n-1}^2 + \frac{1}{n}(x_n - \overline{x}_{n-1})^2 \quad (2)$$

Here x is n^{th} probability $score$, $\overline{x}_n$ is the mean and s_n^2 is the variance. We then calculate the standard deviation $s_n = \sqrt{s_n^2}$.

Now if $\overline{x}_n^b$ and s_n^b are respectively the mean and standard deviation of probability scores of only benign URLs for a particular i. Then for a 95% confidence interval we caculate the benign thresholds as:

$$\epsilon_b = \overline{x}_n^b - 2 \times s_n^b \quad (3)$$

Similarly, for this particular i the malicious threshold is:

$$\epsilon_m = \overline{x}_n^m + 2 \times s_n^m \quad (4)$$

We maintain a record of these parameters for each i in our system.

The above equations depend on the parameter confidence interval (95% in our example). We can decrease the confidence interval to achieve better performance or vice versa. This is because increasing the confidence interval means we will have to collect the more expensive features (Whois and GeoIP) more often.

For example, in our tests we saw that it takes about 4060 ms to collect GeoIP features in contrast to only 3 ms for lexical features. Thus increasing the confidence interval has significant impact upon the system's performance.

IV. Experimental Verification and Result

A. Dataset

We have collected a dataset of about 60000 benign URLs and 38000 phishing URLs for this work. All these URLs are collected from Phishtank.com [25]. Many previous such works used web crawling to generate benign URLs out of highly ranked websites only. This approach often results in a biased dataset because the benign URLs are only sampled from a small subset of all URLs.

Our dataset on the other hand include only verified benign URLs by Phishtank users. The advantage of using this dataset is that these URLs were once reported by a suspecting user as possible phishing URLs. Later other users of Phishtank analyzed and verified them as not malicious. As a result these benign URLs are a better representation of the real world dataset with more common characteristics to phishing URLs than randomly crawled reputed URLs.

Furthermore, for a URL to be valid phishing URL it must host a web-form of some sort to collect user data including username and password. Randomly crawled websites will often not have a sign-in or other such forms. Additionally, sign in URLs often include words like 'login', 'signin' etc. in the actual URL. Other random URLs will not have these words in the URL text, creating noises in the dataset. Our dataset does not suffer from such noise or biases.

TABLE I
Comparison of Batch Learning Classifiers

Algorithm	Accuracy (%)	ROC AUC (%)	Time (s)
KNN	85.12	83.78	0.128
ADB	80.45	77.87	3.205
GDB	83.71	80.68	8.882
DT	84.72	84.19	0.666
RF	**90.51**	**90.35**	1.091
GNB	66.51	58.32	**0.039**
LD	78.08	74.79	0.335
QD	50.29	59.59	0.175
SVC	64.23	51.03	937.4
NuSVC	81.20	74.51	1578

Note: bold values show the best value in a column.

B. Comparison of Batch Learning Algorithms

We begin our experimentation with a comparison of 10 popular 'batch' classification algorithms - 1) K-Nearest Neighbor (KNN) 2) AdaBoost (ADB) 3) GradientBoost (GDB) 4) Decision Tree (DT) 5) Random Forest (RF) 6) Gaussian Naive Bayes (GNB) 7) Linear Discriminant Analysis (LD) 8) Quadratic Discriminant Analysis (QDA) 9) C-Support Vector (SVC) 10) Nu-Support Vector (NuSVC) using scikit-learn [26] a Python [27] library.

The results are summarized in table I. The results clearly show that Random Forest (RF) outperforms all other classifiers for our dataset in reasonable time duration. The time duration depends on the machine and should be considered for relative comparison only.

Random Forest (RF) works best on our dataset primarily because the dataset is sparse, with many missing feature values. RF offers several other advantages for our particular problem. Firstly, their are many categorical features (36 in number) in our dataset. We do not need to use 'one-hot encoding' with RF for the categorical features. We save considerable processing resource by not 'one-hot encoding' the categorical features. Secondly, RF works equally well with unscaled (normalized) and scaled features. This property of RF also saves us significant processing resource. Finally, since RF is somewhat impervious to missing features (-1 in our dataset), we can use the same classifier even after getting new feature sets.

However, we cannot use the batch RF in our framework because of the unbounded growth of our dataset. Rather, we compare these benchmark results with different online learning algorithms in IV-C.

C. Comparison of Online Learning Algorithms

Next, we compare 6 popular 'online' classification algorithms: 1) Perceptron (PE) 2) Stochastic Gradient Descent (SGD) 3) Passive Aggressive (PA) 4) Mondrian Tree (MDT) 5) Mondrian Forest (MDF) 6) Multi-Layer Perceptron (MLP). For Mondrian Tree and Mondrian Forest we used scikit-garden

TABLE II
COMPARISON OF ONLINE LEARNING CLASSIFIERS

Algorithm	Accuracy (%)	ROC AUC (%)
PE	67.17	59.04
SGD	37.10	50.37
PA	63.45	49.98
MDT	82.79	82.42
MDF	**87.47**	**86.63**
MLP	80.77	77.65

Note: bold values show the best value in a column.

[28] and for others we used scikit-learn [26]. The results are summarized in table II.

Mondrian Forest [29] is an online random forest algorithm. It has comparable accuracy to batch random forest algorithm while being remarkably faster than it. Moreover, it generates better results with smaller dataset than other online random forest algorithms [29]. The performance vs accuracy tradeoff offered by Mondrian Forests is suitable for our system.

We do not compare the time taken by each algorithm because an online learning algorithm will be trained per sample and not in batches. The results in table II show that Mondrian Forest (MDF), performs better than all other online learning classifiers and has an accuracy score only second to batch RF.

D. Feature Importance Analysis

We started experimentation with 142 features in total. Then we used the 'drop-column' method to calculate the importance of each feature. In this method, we calculate the importance of a particular feature, by noting the change in accuracy score after removing that feature from the dataset.

The 20 most important features in our dataset are shown in figure 2. It is interesting to note that entropy of the URL string is the most important feature with a contribution of 2.75% in figure 2. This is because malicious URLs often have random characters instead of dictionary words. As a result the entropy of those strings are higher.

The second most important feature, that we found, is the number of days since the domain registration has been updated. As malicious websites are often newer than benign websites, the recently updated domains tend to be malicious. The third most important feature is the maximum number of consecutive digits in URL path. The argument behind this is the same as the one for URL entropy. A random URL string usually has more digits than a regular string made up of words.

A key takeaway from figure 2 is that half of the top 20 features are lexical features. Lexical features are much less expensive to collect, making our delayed feature collection algorithm successful.

E. Delayed Feature Collection

The performance improvement of our delayed feature collection module depends on the *cost* of collecting each feature set. We have calculated the average cost of collecting lexical features, host features, Whois features and GeoIP features to be 3 ms, 155 ms, 1453 ms and 4060 ms respectively. Therefore, it takes our system $3ms$ on average to collect lexical features of an URL and so on. So, if our system can label an URL using only lexical features it can save $4060 + 1453 + 155 = 5668$ ms of time.

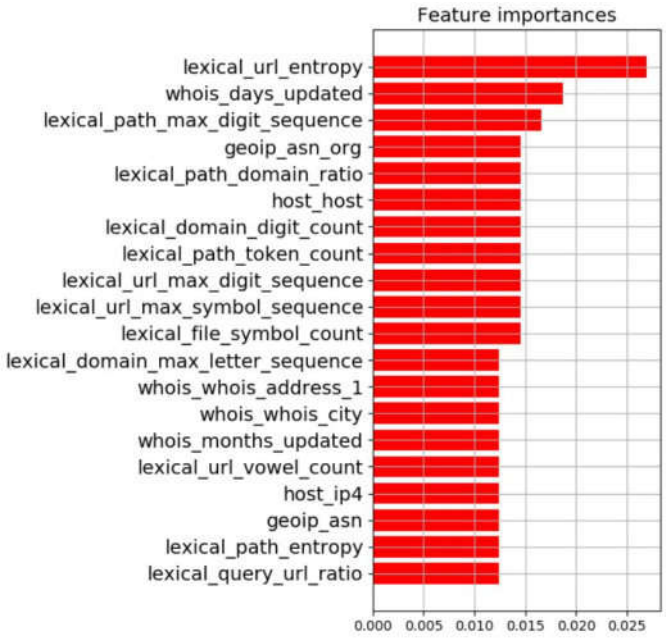

Fig. 2. Top 20 Important Features

Our simulation results show that our system potentially spends about 23% less time in collecting features because of this algorithm. However, a portion of this saved time is spent in repeatedly predicting the probability of this URL. Moreover, this work is still in progress and we are working on improving this algorithm.

F. Cross Validation

To cross validate our findings we have used another dataset of phishing URLs downloaded from OpenPhish.com [30]. We removed our training URLs from that dataset and ended up with 2664 URLs previously not seen by our classifier. Our classifier achieved an accuracy of 86.6% validating that our training set was not biased.

G. Complexity and Scalability

Figure 3 shows that the complexity of our URL classification system is linear. The jumps in the graph are attributed to offline URLs. Some, of the feature collection modules wait for timeout when an URL is offline. We can see two such timeouts at around 200s and 500s. Despite the noises, the graph is linear on average.

Since the process is linear and the detection modules are independent, our system is horizontally scalable. We can scale up our system depending on the traffic.

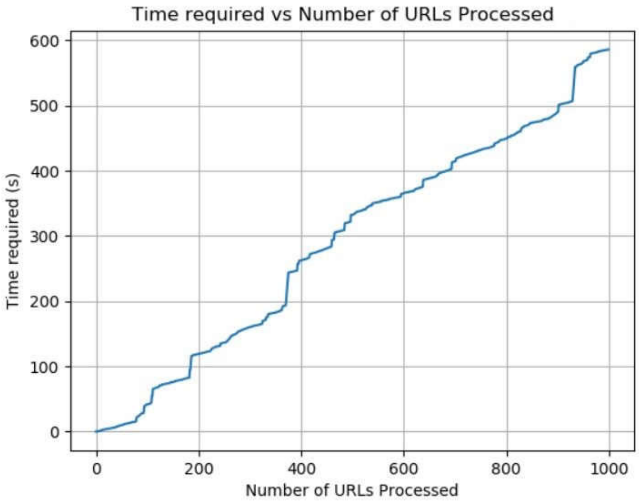

Fig. 3. Linear Complexity of System

V. Conclusion and Future Work

In this work we have outlined a robust framework for automated detection of phishing URLs. We have used online learning to deal with the unbounded growth of URL space. We have also incorporated selective sampling and delayed feature collection to significantly improve the system performance. Our system can detect previously unseen URLs with 87% accuracy.

This work is currently in progress. We are currently working on collecting more feature sets including n-gram, DNS query results, black list presence, bag of words, web page content, web page network traffic etc. We are also working on collecting more URLs to decrease the variance of the probability scores. We are also working on the selective sampling algorithm and unbalanced datasets. Finally, in future we want to include time varying features of the URLs.

References

[1] J. Lewis, "Economic Impact of Cybercrime, No Slowing Down." McAfee, 2018.

[2] T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer, "Social phishing," *Communications of the ACM*, vol. 50, no. 10, pp. 94–100, 2007.

[3] "Enterprise Phishing Resiliency and Defense Report," PhishMe, 2017.

[4] D. Sahoo, C. Liu, and S. C. Hoi, "Malicious url detection using machine learning: a survey," *arXiv preprint arXiv:1701.07179*, 2017.

[5] "Quarterly Threat Trends," https://www.webroot.com/us/en/business/-resources/threat-trends/june-2019/, 2019, [Online; accessed 1-September-2019].

[6] H. Le, Q. Pham, D. Sahoo, and S. C. Hoi, "Urlnet: learning a url representation with deep learning for malicious url detection," *arXiv preprint arXiv:1802.03162*, 2018.

[7] Y.-L. Zhang, L. Li, J. Zhou, X. Li, Y. Liu, Y. Zhang, and Z.-H. Zhou, "Poster: A pu learning based system for potential malicious url detection," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 2599–2601.

[8] R. Verma and A. Das, "What's in a url: Fast feature extraction and malicious url detection," in *Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytics*. ACM, 2017, pp. 55–63.

[9] J. Jiang, J. Chen, K.-K. R. Choo, C. Liu, K. Liu, M. Yu, and Y. Wang, "A deep learning based online malicious url and dns detection scheme," in *International Conference on Security and Privacy in Communication Systems*. Springer, 2017, pp. 438–448.

[10] A. Astorino, A. Chiarello, M. Gaudioso, and A. Piccolo, "Malicious url detection via spherical classification," *Neural Computing and Applications*, vol. 28, no. 1, pp. 699–705, 2017.

[11] A. Blum, B. Wardman, T. Solorio, and G. Warner, "Lexical feature based phishing url detection using online learning," in *Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security*. ACM, 2010, pp. 54–60.

[12] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond blacklists: learning to detect malicious web sites from suspicious urls," in *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2009, pp. 1245–1254.

[13] G. Xiang, J. Hong, C. P. Rose, and L. Cranor, "Cantina+: A feature-rich machine learning framework for detecting phishing web sites," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 2, p. 21, 2011.

[14] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Learning to detect malicious urls," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, p. 30, 2011.

[15] G. Tan, P. Zhang, Q. Liu, X. Liu, C. Zhu, and L. Guo, "Malfilter: A lightweight real-time malicious url filtering system in large-scale networks," in *2018 IEEE ISPA/IUCC/BDCloud/SocialCom/SustainCom*. IEEE, 2018, pp. 565–571.

[16] O. K. Sahingoz, E. Buber, O. Demir, and B. Diri, "Machine learning based phishing detection from urls," *Expert Systems with Applications*, vol. 117, pp. 345–357, 2019.

[17] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[18] M. S. I. Mamun, M. A. Rathore, A. H. Lashkari, N. Stakhanova, and A. A. Ghorbani, "Detecting malicious urls using lexical analysis," in *International Conference on Network and System Security*. Springer, 2016, pp. 467–482.

[19] "2019 Threat Report," https://www-cdn.webroot.com/9315/5113/6179/2019_Webroot_Threat_Report_US_Online.pdf, Webroot, 2019.

[20] P. Zhao and S. C. Hoi, "Cost-sensitive online active learning with application to malicious url detection," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 919–927.

[21] S. C. Jeeva and E. B. Rajsingh, "Intelligent phishing url detection using association rule mining," *Human-centric Computing and Information Sciences*, vol. 6, no. 1, p. 10, 2016.

[22] S. Hollenbeck and A. Newton, "Registration data access protocol (rdap) query format," Internet Requests for Comments, IETF, RFC 7842, March 2015. [Online]. Available: https://tools.ietf.org/html/rfc7482

[23] "GeoLite2 Free Downloadable Databases," https://dev.maxmind.com/geoip/geoip2/geolite2/, [Online; accessed 19-September-2019].

[24] T. Anderson, *The theory and practice of online learning*. Athabasca University Press, 2008.

[25] "Phishtank," http://www.phishtank.com, accessed: 2019-10-30.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[27] G. Van Rossum and F. L. Drake Jr, *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[28] "Scikit-garden: A garden for scikit-learn compatible trees ." [Online]. Available: https://github.com/scikit-garden

[29] B. Lakshminarayanan, D. M. Roy, and Y. W. Teh, "Mondrian forests: Efficient online random forests," in *Advances in neural information processing systems*, 2014, pp. 3140–3148.

[30] "OpenPhish," http://www.openphish.com, accessed: 2019-11-22.