Phishing URL Detection with Oversampling based on Text Generative Adversarial Networks

Ankesh Anand*
Montreal Institute for Learning Algorithms
Montreal, QC, Canada
ankesh.anand@umontreal.ca

Kshitij Gorde*
University of North Carolina
Charlotte, NC, USA
kgorde@uncc.edu

Joel Ruben Antony Moniz*
Carnegie Mellon University
Pittsburgh, PA, USA
jrmoniz@andrew.cmu.edu

Noseong Park†
George Mason University
Fairfax, VA, USA
npark9@gmu.edu

Tanmoy Chakraborty
IIIT-Delhi
New Delhi, India
tanmoy@iiitd.ac.in

Bei-Tseng Chu
University of North Carolina
Charlotte, NC, USA
billchu@uncc.edu

***Abstract*— The problem of imbalanced classes arises frequently in binary classification tasks. If one class outnumbers another, trained classifiers become heavily biased towards the majority class. For phishing URL detection, it is very natural that the number of collected benign URLs (i.e., the majority class) is much larger than the number of collected phishy URLs (i.e., the minority class). Oversampling the minority class can be a powerful tool to overcome this situation. However, existing methods perform the oversampling task in the feature space where the original data format is removed and URLs are succinctly represented by vectors. These methods are successful only if feature definitions are correct and the dataset is diverse and not too sparse. In this paper, we propose an oversampling technique in the data space. We train text generative adversarial networks (text-GANs) with URLs in the minority class and generate synthetic URLs that can be made part of the training set. We crawl a crowd-sourced URL repository to collect recently discovered phishy and benign URLs. Our experiments demonstrate significant performance improvements after using the proposed oversampling technique. Interestingly, some of the original test URLs are exactly regenerated by the proposed text generative model.**

***Keywords*-phishing; text-GANs; generative adversarial networks; oversampling**

I. Introduction

Cyberattacks are a serious concern to any organization. These attacks create huge societal and financial losses worldwide. In many cases, phishing is a starting point of these cyberattacks. Careless users who follow phishing emails and links can be infected by malicious code, allowing the attacker to intrude into the enterprise network though the infected machines.

Prior work has demonstrated that the use of machine learning to detect such phishing URLs is promising. Several methods have been proposed to tackle this problem. These methods can be categorized broadly into two types: content-based [24] and URL-based [15], [16], [19], [18], [23], [3]. Content-based methods analyze the contents of phishing web-pages; whereas URL-based methods simply check their URL string patterns. It has been shown that URL-based methods show comparable performance to content-based methods [4], [6]. More importantly, there are no security risks involved in URL-based methods because they do not download malicious web-pages that may exploit browser-based vulnerabilities.

However, there exists an important problem intrinsic to this domain: *imbalanced datasets*. In practice, benign URLs outnumber phishy URLs. We practically observe this problem in the datasets used in this paper as well (for example, the number of benign URLs is twice that of phishy URLs, as reported in Table I). Classifiers trained with such imbalanced datasets often produce predictions biased towards the majority class, and are likely to be reluctant to predict a URL as phishy.

In general, this situation is often tackled by oversampling the minority class[1], and several methods have been proposed for oversampling in the feature space [5], [11], [12], [20]. Existing feature-space oversampling methods create synthetic samples which are similar to real samples by i) converting data samples into vectors of a feature space, and ii) finding randomly weighted combinations of existing vectors.

In this work, we propose the novel concept of *oversampling in the data space*. We train generative adversarial networks (GANs) [9] to learn about the string pattern statistics of URLs in the minority class, and use them to

*Equally contributed and listed in alphabetical order; †Corresponding author; This work was partially supported by the Office of Naval Research under the MURI grant N00014-18-1-2670, and the Indo-UK Collaborative Project DST/INT/UKP158/2017.

[1]Another option is to undersample the majority class, which means dropping benign URLs, in spite of the effort spent to collect them. This reduces the training size and is preferred when the training set is overpopulated, which is not the case in our paper.

978-1-5386-5035-6/18/$31.00 ©2018 IEEE

generate synthetic URLs. We observe that these synthetic URLs cannot be generated by conventional feature-space oversampling methods. A very large number of synthetic URLs are generated by the GAN, all of which cannot be added to the minority class. Thus, we employ techniques to select representative synthetic samples (using k-Means clustering and Euclidean distance-based selection). The minority class is augmented with the so selected representatives to generate equal-sized classes.

We also collected URLs from phishtank.com over a period of six months, since existing datasets either contain only URL feature vectors without text or contain outdated phishing URLs [15], [16], [24], [18] (although we show that our oversampling method is effective in these old datasets as well). This dataset of collected URLs was subsequently divided into an 80-20 train-test split. To test the efficacy of our proposed oversampling technique, we perform experiments with several state-of-the-art classifiers. In all cases, our method outperforms other oversampling methods by significant margins.

It is observed that around 40$\sim$80 phishing URLs in the test set are generated by the proposed method, which shows the possibility of hunting phishing URLs (since detection is sometimes too late).[2]

II. Related Work

A. Phishing URL Detection

Extensive work has been done to counter phishing attacks, for example, [15], [16], [4], [19], [18], [8], [23] and [3]. Typically researchers have explored machine learning techniques to automatically detect phishing URLs. It is vital to have a well-defined set of features for the effectiveness of classification algorithms.

The objective of this paper is not to propose new features: existing features sufficiently serve that purpose. This section serves to review various features presented in existing work, based on which the oversampling technique proposed in this paper is tested.

In [19], the authors propose several features related to phishing websites by studying 2500 phishing URLs. The authors categorize the features into four categories: address bar features; abnormality-based features; HTML and JavaScript-based features; and domain-based features. [4] proposes using only lexical features to detect phishing URLs. There also exist many unsupervised approaches such as blacklisting and website scoring based on reputation [8]. However, these techniques are not reliable since attackers may come up with methods to generate new URLs. In our research, we rely mainly on the lexical features of a string to predict phishing URLs. The following feature definitions are widely used in many of the aforementioned works:

[2]The phishing URLs we hunted are listed in the appendix. However, the recall rate has scope for improvement. This is left as an open problem.

1) *No usage of domain name*
There is a high probability of a URL being phishy if it directly uses IP address instead of domain names to make it hard to recognize the phishing website (for instance, if a URL looks like http://145.145.1.0/signup, then there is a high chance that it is phishy). Alternatively, some phishy sites also have hexadecimal notations in place of an IP address such as http://0xC0.0xA8.0x2A.0x48.
2) *URL Length*
The length of the URL is a significant factor in detecting phishy URLs. Attackers try to use lengthy URLs to conceal the appearance of underlying phishy URLs. We use the same length factors as described in [19] as follows:

$$\text{A URL is } \begin{cases} \text{benign, if its length} \leq 53 \\ \text{neutral, if } 54 \leq \text{its length} \leq 75 \\ \text{phishy, if its length} \geq 76 \end{cases} \quad (1)$$

3) *Domain separated by "-'*
Adding prefix/suffix separated by "-" is a frequent indicator of an illegitimate website. Attackers frequently use this technique to lure users to the phishy website by exploiting an untrained eye. For instance, an attacker may use Microsoft's domain separated by a prefix as "http://www.microsoft-sharepoint-login.com".
4) *Multiple sub-domains*
[19] stated a factor to consider a url to be phishy based on the number of sub-domains incorportated in the url.Thus, if a URL's resource name part has more than three sub-domains (indicated by dots), then there is a high chance that it would be a phishy URL. An example of such a URL would be "http://www.google.3432win.au".
5) *Usage of "@" symbol*
Internet browsers ignore everyting that comes before "@" symbol. Attackers tend to use this trick to propoagate the phishy URLs. Attackers employ this trick to take advantage of an untrained eye. For eg: an attacker can use the address "http://www.xbox.com@tactical.com" which causes the browser to ignore "www.xbox.com" and proceed to "tactical.com".
6) *Count of TLD (Top-level domains) in the path*
Adding multiple top-level domains in the URL's path is a smart move to impersonate legitimate websites. This feature is turned on when the number of top-level domains in the URL exceeds one.
7) *Counting suspicious words*
We keep a track of some of the suspicious and common words occuring in the phishing world to train our models. After analyzing various datasets we randomly choose some suspicious words like "login",

"account", "credit", "secure", "update" as well as system commands like "mkdir", "cmd" and "admin".

8) *Number of punctuation symbols used*
We maintain a count for common punctuation symbols such as .!&,#$%;. [23] observed a high percentage of punctuation symbols in phishy URLs thus making it an important factor in distinguishing phishy urls from legitimate ones
9) *Digits in the domain name*
A legit URL would not have any digits in the domain. If a digit is detected in the URL, there is a strong change of the url being phishy. This feature is turned on if digits are detected in the domain name of the URL.
10) *Entropy*
[3] found that entropy as a feature is highly significant. Legitimate URLs always have meaningful names causing their entropies to be lower than phishy URLs which tend to have gibberish text.
11) *Kullback-Leibler (KL) divergence*
The Kullback-Leibler (KL) divergence is a metric to measure the similarity between two probability distributions. Using the reference for the character distribution of the English language, we can compare the similarity between URLs character distribution with that of the English language. The character distribution of the English language is obtained from [17].
12) *Counting "-" in the path*
Legit URLs tend to not have multiple "-" in the path. Hence the count of "-" in the path is another important feature.
13) *Vowel/Consonant ratio*
VC ration indicates the ratio of total vowels to total consonants in the hostname part of the URL. The VC ratio for phishy URLs tend to highly differ from legitimate URLs.
14) *Digit/Letter ratio*
This feature calculates the ration of digits to the letters in the URL.
15) *Usage of Brand names*
Phishy URLs generally try to mimic some of the top visited website brands on the internet. Thus, we created a list of 1000 most visited websites from Alexa and used them as popular brands. Phishy URLs tend to use these top brand names with additional prefixes or suffixes such as "xbox-xyz.com", "google-g.com", "wsj-st.com" etc.
16) *Long hostnames*
Legitimate URLs tend to have short hostnames. Conversely, phishy URLs tend to have very long hostnames. This feature is activated if the length of the hostnames exceeds 22 characters.
17) *Very short hostnames*
Legitmate URL has the right balance for the length of the URL. Thus, if the hostname is extremly short (fewer than 5 characters), then that could be a potential indicator of maliciousness.
18) *Count ":" in the hostname*
Port number manipulation is attempted by some phishy URLs and a great indicator for such is the presence of ":" in the URL.

B. Oversampling

Oversampling is a classical yet powerful technique to deal with imbalanced datasets. It tries to match the size of two classes by adding artificial samples to the minority class. Many previous works have studied this problem. Among them, we consider the following 6 reliable methods in this paper: Random minority oversampling with replacement (RMR), Synthetic minority over-sampling (SMOTE) [5], Borderline SMOTE Type 1 (bSMOTE1) [11], Borderline SMOTE Type 2 (bSMOTE2) [11], Support vector SMOTE (SVMSMOTE) [20], Adaptive synthetic sampling (ADASYN) [12].

All these methods perform the task in the feature space (an 18-dimensional vector space in our case, because there are 18 features) where each URL is represented by a vector. SMOTE finds several nearest neighbor vectors from a randomly chosen vector in the feature space, and calculates a vector that is their randomly weighted-average as a synthetic sample. As SMOTE decides weights randomly in [0,1], it actually produces purely random combinations of existing samples without considering other factors. ADASYN was designed on top of SMOTE, where weights are adjusted by accounting for the difficulty of the minority class samples in learning.

One problem of existing feature-space oversampling methods is that synthetic vectors are confined to combinations of existing vectors (for example, randomly weighted-average vectors in SMOTE). Thus, they cannot perform well if existing samples are sparse or lack diversity in their vector representations.

Our proposed method performs the oversampling task in the data space (i.e., oversampling with the original raw text data). Synthetic URLs in our method have the same statistical string pattern distribution as that of the URLs in the minority class and in many cases, those synthetic URLs cannot be represented by combinations of existing vectors in the feature space.

Therefore, our approach based on text-GANs has a clear advantage over classical oversampling techniques, that is, our method produces semantically meaningful synthetic samples rather than random combinations of existing samples.

C. Generative Adversarial Networks

Generative adversarial networks (GANs) constitute a novel framework for estimating generative models via an

Input: Real Samples: $\{x_1, x_2, \cdots\} \sim p(x)$
Output: a Generative Model G

$G \leftarrow$ a generative neural network
$D \leftarrow$ a discriminator neural network
while *loss values have not converged* **do**
    Create a mini-batch of real samples $X = \{x_1, \cdots, x_n\}$
    Create a set of latent vector inputs $Z = \{z_1, \cdots, z_n\}$
    Train the discriminator D by maximizing Equation (2)
    Train the generator G by minimizing Equation (2);
end
return G

Algorithm 1: Training algorithm of GANs

adversarial process, in which a generative model G and a discriminative model D (for example, both neural networks) are trained simultaneously. The general idea is analogous to two models being pitted against each other, where one model counterfeits (for example) data and the other model estimates the probability of whether the data given to it is fake or not. This competition ideally drives both models to improve until the generated data samples become indistinguishable from real ones. When a generative model is run, $G = G(z) \sim p_g$ receives random noise $z \sim p_z$ as input. After an optimal training procedure, p_g can be shown to match p_{data}, the distribution governing the real data samples x [9]. The provably optimal training procedure is cast in the form of a two-player objective:

$$\min_G \max_D \quad \mathbb{E}_{x \sim p_{data}(x)}[log(D(x))] + \mathbb{E}_{z \sim p_z(z)}[log(1 - D(G(z)))] \tag{2}$$

Algorithm 1 shows the general training algorithm of GANs. G and D can be any form of neural networks. The discriminator D attempts to maximize the objective, whereas the generator G attempts to minimize it. In other words, the discriminator D attempts to distinguish between real and generated samples, while the generator G attempts to generate realistic fake samples that the discriminator D cannot distinguish from real samples. One can also consider the discriminator as a teacher and the generator as a student, where the teacher provides feedback to the student on the quality of work.

III. Problem Statement

In this paper, the primary focus is on URL-based detection techniques. The authors of [4], [6] demonstrated that with an appropriate training set, a URL-based approach can achieve a high degree of accuracy for classifying previously unseen phishing URLs. This indicates that a model operating solely on features derived by the inspection of URL strings performs similar to content-based detection systems, eliminating the costs and security risks associated with such systems. **In addition to runtime overheads when dealing with content-based detection techniques, there is an inherent risk of being exposed to browser-based vulnerabilities which needs to be avoided.** Moreover, it is in the best interest of the user to avoid downloading content from phishy websites. Further, reliably obtaining phishy websites for training and testing purposes can be difficult. Focusing on URL-based detection makes the system general enough to be applicable to any context, such as emails, chats and web-pages. URL-based classification is also a lightweight process compared to the other categories.

We do not propose new features for URLs, but use well-known existing features. Our goal is to optimize the training data by adopting advanced machine learning techniques to achieve better classification accuracies. One of the most common problems in classification is an imbalanced dataset. In such datasets, the data samples in one class significantly outnumber those in the other. The minority class is generally the most significant in phishing URL detection, since it represents the concept to be learned. A model trained on such imbalanced datasets is likely to be biased and inaccurate, since machine learning algorithms are typically trained by reducing errors or by minimizing a loss function. Various challenges need to be overcome to effectively use classification algorithms while dealing with imbalanced datasets. These algorithms tend to discard small clusters of the minority class as noise, but still show a high accuracy by only predicting the majority class.

Here, we solve the imbalanced dataset problem by designing an advanced oversampling technique based on text-GANs.

IV. Oversampling based on text-GANs

In this section, we describe our proposed oversampling technique and GAN model in detail. The class imbalance problem discussed above is solved as follows:

1) A GAN model is trained with the minority class in our dataset.
2) The trained GAN model is used to generate several URLs.
3) A subset comprising of high-quality generated URLs is chosen by representative selection methods, and this subset is used to augment the dataset's minority class.
4) If the number of representatives obtained after Step 3 is insufficient, existing feature-space oversampling algorithms are run to achieve a balance between the two classes.
5) With the now completely balanced dataset, state-of-the-art classification algorithms are trained to predict phishing URLs.

Usually, the GAN model in Step 1 generates a large number of candidate URLs. All of these URLs cannot be used to augment the minority class because this creates an imbalanced situation yet again, now in the opposite direction. Instead, a subset of representative URLs representative of those generated in Step 2 is carefully chosen, for which we propose three representative selection methods in Section IV-B. After adding the selected representatives into the

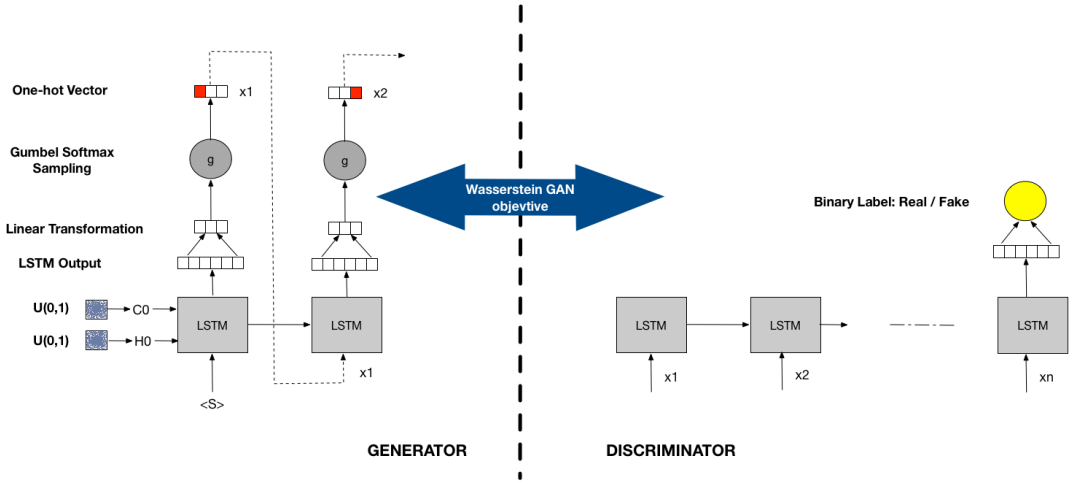


Figure 1: A schematic diagram of our GAN model architecture. We use a Wassertein GAN objective along with Gumbel-Softmax sampling. The Gumbel-Softmax sampling is differentiable; thus, training the proposed GAN does not require REINFORCE, unlike the work in [25].

minority class, we perform existing oversampling methods in the feature space to achieve perfect balance if the number of generated representatives is too small to solve the imbalance. Our experiments show that the proposed combination of representative selection based on text-GANs and existing feature-space oversampling significantly improves predictive accuracy. Recall that existing feature-space oversampling methods produce merely weighted combinations of existing vectors. After strengthening the minority class with synthetic samples that cannot be synthesized by existing methods, the samples produced are more diverse than when relying solely on existing feature-space oversampling methods.

A. Text Generative Adversarial Networks (text-GANs)

Although several methods have been proposed to improve GANs, it is well-known that their training is often unstable [1]. In particular, GANs for generating texts are much harder to train than GANs for images [25]. Therefore, we adopt two recently proposed techniques: Wasserstein metric [2] and Gumbel Softmax [14] are adopted. Figure 1 shows a schematic diagram of the GAN architecture used in this paper.

1) Wasserstein Objective:: The Wasserstein metric measures the distance between two probability distributions as follows:

$$W(p_z, p_{data}) = \sup_{\|f\|_L \leq K} \mathbb{E}_{x \sim p_{data}}[f(x)] - \mathbb{E}_{z \sim p_z(z)}[f(z)], \quad (3)$$

where the supremum is over all the K-Lipschitz functions. [2] noted that a neural network is able to roughly approximate such K-Lipschitz functions by clamping its weights to a fixed box after each gradient update.

In order to improve the stability of the GAN training process, we adopt the Wasserstein GAN training objective. We also incorporate a gradient penalty term, which gets rid of undesirable behaviours due to weight clipping in the original Wasserstein GAN implementation [10]. The new objective to train the discriminator then becomes:

$$L = \mathop{\mathbb{E}}_{z \sim p_z(z)}[D(G(z))] - \mathop{\mathbb{E}}_{x \sim p_{data}(x)}[D(x)] + \lambda \mathop{\mathbb{E}}_{\hat{x} \sim p_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2], \quad (4)$$

where $\hat{x}$ represents straight lines between the data distribution and the generator distribution, and λ is a hyperparameter which controls the trade-off between optimizing the penalty term and the original Wasserstein objective.

2) Recurrent Architecture:: Both the generator and the discriminator in our model are LSTMs [13]. The generator generates a sequence character-by-character, where each character is represented through a one-hot encoding. The cell state and the hidden state of the generator are initialized by a pair of vectors, sampled from $z \sim U(0,1)^d$.

At each time-step of the LSTM, the LSTM output is passed through a linear layer followed by a Gumbel-Softmax layer. The Gumbel-Softmax layer samples from the Gumbel-Softmax distribution and produces a discretized output (described in detail below). The character generated at a given time-step is then fed into the LSTM as the next timestep's input.

3) Gumbel-Softmax Distribution:: The Gumbel-Softmax distribution can be leveraged to train neural networks with discrete latent variables. Specifically, the Gumbel-Softmax trick provides an efficient way to draw samples z from a

URL Example 1: Real phishing URLs

1. http://www.51jianli.cn/images?ref=http:./pxzugfbus.battle.net/d3/en/index
2. http://autobilan−chasseneuillais.fr/bretmrae/garyspeed/cc8aa05c4241840f
3. http://siliconebands.ca/wp−content/Gdocs/d35fca89d8835604fece02
4. http://bermongkol879.com/css/AA/e7f9a7249d0580cb6b85090a60594693

URL Example 2: Phishing URLs generated by our model

1. http://www.51jianli.com/Adobe/34fd80f5a07514376723e01188/index.php
2. http://www.autobilan−chasseneuillais.com/com/53dfcafe2230b16749
3. http://www.siliconebands.teewhy/wp−includes/9da7a
4. http://www.bermongkol879.com/themes/cdf0bfe5391993b89c177e66485ceb39

Figure 2: Real and synthetic URL samples

categorical distribution with class probabilities π_i:

$$z = \mathbf{onehot}(arg\ \underset{i}{max}[g_i + log\pi_i]), \tag{5}$$

where g_i are i.i.d samples drawn from the Gumbel distribution with zero mean and unit variance. Since the $argmax$ function is not differentiable, we simply use the softmax function as a continuous approximation of $argmax$:

$$y_i = \frac{exp(log((\pi_i + g_i)/\tau)}{\sum_{j=1} exp(log((\pi_j + g_j)/\tau)}, \tag{6}$$

where τ is a temperature parameter that allows us to control how closely samples from the Gumbel-Softmax distribution approximate those from the categorical distribution.

As $\tau \rightarrow 0$, the softmax becomes an argmax and the Gumbel-Softmax distribution becomes the categorical distribution. During training, we let $\tau > 0$ to allow gradients past the sample, then gradually anneal the temperature τ (though not completely to 0, as the gradients would then blow up).

B. Representative Selection

Among many generated URLs, we select a small set of representative ones to be part of the training set. In our experiments, we generate millions of URLs and select thousands of representative ones on average.

1) k-Means Clustering:: Each generated URL is projected onto the feature space, where we run the k-Means clustering algorithm to find k centroids (i.e., representatives). A crucial part of this method is the choice of the parameter k. We use an *unsupervised metric* called the *silhouette score* [21] for this purpose. The silhouette score calculates the consistency of clusters without ground-truth cluster labels. As similar URLs are assigned to a cluster, its silhouette score increases. It is formally defined as:

$$silhouette\_score(e) = \frac{b(e) - a(e)}{\max(a(e), b(e))},$$

where e is a generated URL sample, $a(e)$ is the average distance between e and other URLs of the cluster it belongs to, and $b(e)$ is the minimum average distance between e and other URLs of the clusters it does not belong to.

By definition, it prefers small clusters and becomes 1 if all clusters are singleton clusters (i.e., each generated URL constitutes its singleton cluster.) Because of this, we use the value k where the average silhouette score of all generated samples is saturated. This is called the *elbow method* because the average silhouette score curve (that monotonically increases) looks like a bent arm and the first saturation point is called *elbow joint*. For instance, as k increases, the silhouette score also increases and its elbow joint is the value of k where the silhouette score curve starts to saturate. In other words, adding one more cluster does not greatly improve the clustering performance after the elbow joint.

In general, k is not large enough to match the size of the two classes. Thus, we run a feature-space oversampling algorithm to completely match the size after choosing k representative generated URLs. Because of the additional k URL samples, the feature-space oversampling shows very different behaviors compared to the case when it is executed without them. Note that the samples generated by feature-space oversampling without the k URLs (shown in Figure 3(b)) are not as diverse as those generated when the k URLs are included (shown in Figure 3(d)).

2) Euclidean Distance:: In the feature space, we first find the centroid (i.e., the mean vector) of the minority class, and choose k generated URLs whose Euclidean distances from the centroid are the largest. Thus, this Euclidean distance method prefers samples different from the ones existing in the minority class. Sometimes, this generates better prediction results than the k-Means clustering method because of the increased diversity of the chosen URLs.

In this method, we set $k = \alpha \cdot (|Majority| - |Minority|)$

(where k, the number of chosen URLs, would be half of the class size difference when $\alpha = 0.5$, for example). To resolve the remaining imbalance, we run feature-space oversampling as in the previous k-Means clustering method and make the sizes of the two classes equal.

3) Random Selection: We choose $k = \alpha \cdot (|Majority| - |Minority|)$ to generate URLs randomly. This method is tested to show that principled representative selection methods are required. While this method does show better performance than the previous two methods occasionally, we observe that in almost all cases, the two aforementioned methods (namely, clustering-based and distance-based) lead to much more reliable performance in our experiments.

C. Generated URL Examples

We show some samples of real and synthetic URL examples in Figure 2. The synthetic URLs in the second list have similar characteristics to the real ones in the first list.

V. Comparison with Related Work

In this section, we compare selected oversampling examples for our dataset. Figure 3 shows various oversampling results. Our feature space is 18-dimensional and for visualization, Hessian Eigenmapping [7], a manifold learning algorithm, is applied to project the vectors into 2-dimensional space. Figure 3(a) shows vectors (points) of phishing URLs for PayPal (see Section VI-A) present in our dataset. As expected, many points are located around the center of the class. After running SMOTE, we show the additional generated samples as blue points in Figure 3(b). Note that most blue points are confined to the vicinity of the red points.

Points generated by our GAN are marked in green in Figure 3(c). They are not restricted to regions directly around the red points, and some points that were previously impossible to generate in Figure 3(b) are created (e.g., three green points in the right bottom area). Figure 3(d) shows the result of applying SMOTE to Figure 3(c).

VI. Experiments

In this section, we introduce our dataset and experimental setup, and discuss detailed experimental results to show the efficacy of the proposed oversampling method.

A. Phishing URL Dataset

Attackers' phishing attack patterns quickly evolve based on defenders' methods. This is common for many cyberattacks. It is thus always preferred to test with state-of-the-art attack patterns. With this in mind, we crawled phishtank.com and collected URLs reported during the most recent 6 months for Bank of America, eBay, and PayPal. phishtank.com is a crowd-sourced repository of suspicious URLs that does not provide ground-truth labels — users can upvote or downvote the reported URLs in the website, but the reported URLs are not reliable because anyone (including attackers themselves) can participate.

Table I: The number of phishy and benign URLs for each dataset. Note that Sorio et al.'s data is already tagged with ground-truth labels, and we did not need to use VirusTotal.

Dataset	VirusTotal Threshold	Phishy URLs	Benign URLs
ebay	10	8,529	18,800
PayPal	10	9,690	17,572
Bank of America	10	4,610	9,408
Sorio et al.	N/A	6,564	82,101

Instead, we used virustotal.com to tag collected URLs. This website returns the prediction results of over 60 anti-virus (AV) products given a URL. The 10 most reliable and popular AV products (such as McAfee, Norton, Kaspersky, Avast, Trend Micro, etc.) were selected among these, and consider a URL is considered phishy if all of them indicate so. Table I shows the statistics of the dataset.

We also test on the dataset created by Sorio et al. [22]. While they manually selected a subset of their URLs for each class to create a balanced training set, this might be sub-optimal due to the manual effort required and the smaller number of training samples. Thus, we use their full dataset that is highly imbalanced, as shown in Table I.

B. Experimental Setup for Classification Task

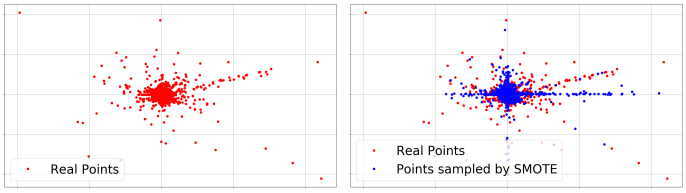
We follow the standard classification task evaluation protocol. A randomly chosen subset comprising of 80% of the phishy and benign URLs is used for training, and the remaining 20% are reserved for testing. For training, we perform grid search and 10-fold cross-validation to choose the best hyper-parameter set. We consider DecisionTree, AdaBoost, RandomForest, SVM, RBM, Multilayer Perceptron, etc. For feature-space oversampling, we consider all 6 methods explained in the related work section. Thus, the proposed method is compared with the following:

1) **Baseline** does not perform any oversampling and trains classifiers with the original dataset;
2) **Oversampling (feature space)** performs existing feature-space oversampling methods and trains classifiers with the augmented dataset. We perform a grid search to find the best oversampling method among available 6 options. In Tables III to V, we report scores along with the best feature-space oversampling method obtained from the grid search.

We use F-1, F-2, and AUCROC score as the main metrics to evaluate the performance. F-1 is the harmonic mean of precision and recall and F-2 gives a more weight on the recall of the phishy class. All tasks were implemented with Python and executed on a cluster of 64 machines running Linux with Xeon 5140 CPU and 24GB RAM.

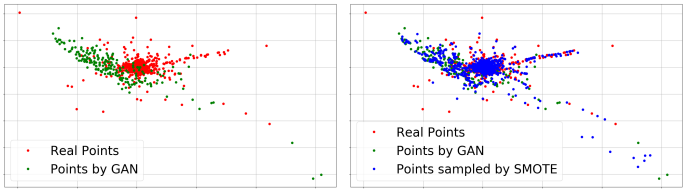
C. Experiment Results

Phishing URL Hunting Results: : Hunting unknown phishing URLs refers to the process of predicting exact URL



(a) Phishing URLs for PayPal in our dataset

(b) SMOTE oversampling



(c) Our proposed text-GAN

(d) Our final proposed method

Figure 3: Real and synthetic URLs in a 2-dimensional space. We used Hessian Eiggenmapping [7], a dimensionality reduction algorithm, to project URLs onto the 2-dim space. (a) Phishing URLs for PayPal in our dataset; (b) Blue points represent synthetic samples by SMOTE; (c) Green points are by the proposed GAN after the Euclidean Distance based representative selection ($\alpha = 0.5$); note that green points show a different distribution from red points because the representation selection method prefers the farthest samples in Euclidean distance; after the dimensionality reduction, they somehow overlap; (d) We applied SMOTE on (c) to obtain the augmented minority class (which comprises of the points of all three colors). The two classes have the same size after (d). For other visualization examples, refer to the appendix.

Table II: The number of successful phishing URL hunting in each dataset. For instance, 82 test phishing URLs of ebay are generated exactly by the proposed model.

ebay	PayPal	Bank of America	Sorio et al.
82	48	37	121

strings and checking their existence in order to block them. Conventional feature-space oversampling techniques do not return strings; therefore only our method can be used for this purpose. Note that this is not our main goal; surprisingly, we find that some test phishing URLs are recreated by our proposed method: 40∼80 phishing URLs in our test sets are recreated during experiments as shown in Table II. For Sorio et al.'s dataset, our method is able to produce over 100 unknown phishing URLs in their test set.

Phishing URL Detection Results: : In Tables III to VI, detailed scores are summarized for each dataset. Without any oversampling (i.e,. Baseline columns in the tables), the F-1 score ranges from 0.38 to 0.55 in the three vendor datasets collected by us, and is 0.89 in the Sorio et al. dataset[3]. All baseline methods achieved F-2 scores less than 0.4, except in the Sorios et al. dataset. This means that many phishing URLs are not identified, indicating unreliability. All classifiers were biased toward the majority class, and were reluctant to classify test URLs as phishy in our dataset.

[3]In fact, it is much easier to detect phishing URLs in the Sorio et al. dataset because their URL strings collected in 2013 are well captured by state-of-the-art lexical features. This also proves that phishing URL patterns evolve to evade detection methods.

Table III: Phishing URL detection results (F-1, F-2, and AUCROC) for the ebay dataset. The best results are indicated in bold font. In all cases, our method outperforms others with non-trivial margins. Our method runs in two stages: i) data-space oversampling with representative selection, and ii) feature-space oversampling if two classes do not have the same size after the data-space oversampling. The name of the feature-space oversampling method that leads to the best performance for the proposed method is indicated in the last column.

Metric	Baseline	Oversampling (Feature Space)	Oversampling (Data+Feature Space, Our method)
F-1	0.3853	0.5095 (bSMOTE2)	k-Means: 0.6863 (RMR)
			Euclidean: **0.6991** (SVMSMOTE)
			Random: 0.6712 (ADASYN)
F-2	0.3065	0.58 (bSMOTE1)	k-Means: **0.6860** (SVMSMOTE)
			Euclidean: 0.6563 (ADASYN)
			Random: 0.67 (RMR)
AUCROC	0.5071	0.5643 (SMOTE)	k-Means: 0.686 (SVMSMOTE)
			Euclidean: 0.6912 (ADASYN)
			Random: **0.701** (bSMOTE2)

Table IV: Phishing URL detection results for PayPal

Metric	Baseline	Oversampling (Feature Space)	Oversampling (Data+Feature Space, Our method)
F-1	0.5512	0.5881 (SVM SMOTE)	k-Means: 0.6889 (SVMSMOTE)
			Euclidean: **0.6993** (ADASYN)
			Random: 0.6919 (ADASYN)
F-2	0.39	0.55 (SMOTE)	k-Means: **0.81** (SVMSMOTE)
			Euclidean: 0.79 (bSMOTE2)
			Random: 0.74 (bSMOTE2)
AUCROC	0.5457	0.58535 (SMOTE)	k-Means: 0.6901 (SVMSMOTE)
			Euclidean: **0.7101** (SVMSMOTE)
			Random: 0.6781 (ADASYN)

Table V: Phishing URL detection results for Back of America

Metric	Baseline	Oversampling (Feature Space)	Oversampling (Data+Feature Space, Our method)
F-1	0.4348	0.5631 (ADASYN)	k-Means: 0.6819 (SVMSMOTE)
			Distance: 0.6950 (bSMOTE2)
			Random: **0.7012** (bSMOTE2)
F-2	0.3984	0.6 (ADASYN)	k-Means: **0.7807** (RMR)
			Euclidean: 0.6962 (SVMSMOTE)
			Random: 0.6829 (ADASYN)
AUCROC	0.5092	0.58 (RMR)	k-Means: 0.6964 (SVMSMOTE)
			Euclidean: **0.701** (SVMSMOTE)
			Random: 0.6860 (ADASYN)

After performing feature-space oversampling, both the F1 and F2 scores are stabilized between 0.5 and 0.6 in our dataset, and between 0.93 and 0.97 in the Sorio et al. dataset. In some cases, it shows much better performance than the previous baseline (for example, from 0.3 to 0.58 in the ebay dataset and from 0.43 to 0.56 in the Bank of America). This demonstrates how conventional oversampling techniques can enhance the classification task.

The proposed method in the last columns shows the best performance in all cases and, in almost all cases, achieves scores that are larger than 0.7. Our method outperforms others by significant margins: for example, our method shows the F-2 score of 0.81 for the PayPal dataset but other baselines remain 0.39 and 0.55 (i.e., 107% and 47% improvements over the two baseline methods, respectively). As expected, representative URL selection based on k-Means or Euclidean Distance outperforms the random representative selection in almost all cases, which supports the importance of the representative selection.

VII. Conclusion

Oversampling is considered as the best method to address the imbalanced dataset problem. However, existing methods, given a set of URLs, find weighted combinations of vectors representing the URLs in the feature space.

We presented a novel method to generate a better set of synthetic URLs, inspired by a recently proposed generative model called Generative Adversarial Networks. We also collected recently discovered suspicious URLs. As attackers revise their URL generation strategy, we wanted to ensure that the dataset is updated in a timely manner. Throughout our experiments, we observed our model yielding significant improvements over baseline methods. We plan to release the dataset and code associated with the paper soon.

Table VI: Phishing URL detection results for the Sorio's dataset

Metric	Baseline	Oversampling (Feature Space)	Oversampling (Data+Feature Space, Our method)
F-1	0.8908	0.9421 (SVMSMOTE)	k-Means: **0.9738** (RMR)
			Distance: 0.9512 (ADASYN)
			Random: 0.9445 (bSMOTE2)
F-2	0.8849	0.9384 (bSMOTE2)	k-Means: **0.9698** (bSMOTE2)
			Euclidean: 0.9501 (RMR)
			Random: 0.9354 (SVMSMOTE)
AUCROC	0.9298	0.9497 (RMR)	k-Means: 0.9365 (bSMOTE2)
			Euclidean: **0.9765** (ADASYN)
			Random: 0.9405 (ADASYN)

References

[1] M. Arjovsky and L. Bottou. Towards principled methods for training generative adversarial networks. *CoRR*, abs/1701.04862, 2017.

[2] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. *ArXiv e-prints*, Jan. 2017.

[3] A. C. Bahnsen, E. C. Bohorquez, S. Villegas, J. Vargas, and F. A. Gonzalez. Classifying phishing urls using recurrent neural networks. In *APWG Symposium on Electronic Crime Research, eCrime*, 2017.

[4] 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*, New York, NY, USA, 2010.

[5] K. W. Bowyer, N. V. Chawla, L. O. Hall, and W. P. Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813, 2011.

[6] M. Darling, G. Heileman, G. Gressel, A. Ashok, and P. Poornachandran. A lexical approach for classifying malicious urls. In *2015 International Conference on High Performance Computing Simulation (HPCS)*, pages 195–202, July 2015.

[7] D. L. Donoho and C. Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. *Proceedings of the National Academy of Sciences*, 100(10), 2003.

[8] M. N. Feroz and S. Mengel. Phishing url detection using url ranking. In *2015 IEEE International Congress on Big Data*, pages 635–638, June 2015.

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[10] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved training of wasserstein gans. *arXiv preprint arXiv:1704.00028*, 2017.

[11] H. Han, W.-Y. Wang, and B.-H. Mao. Borderline-smote: A new over-sampling method in imbalanced data sets learning. In *Proceedings of the International Conference on Advances in Intelligent Computing*, Berlin, Heidelberg, 2005.

[12] H. He, Y. Bai, E. A. Garcia, and S. Li. Adasyn: Adaptive synthetic sampling approach for imbalanced learning. In

IEEE International Joint Conference on Neural Networks, pages 1322–1328, 2008.
[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
[14] E. Jang, S. Gu, and B. Poole. Categorical Reparameterization with Gumbel-Softmax. *ArXiv e-prints*, Nov. 2016.
[15] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Beyond blacklists: Learning to detect malicious web sites from suspicious urls. In *Proceedingsof theSIGKDD Conference. Paris,France*, 2009.
[16] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Identifying suspicious urls: An application of large-scale online learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009.
[17] MEC and Cornell University. https://www.math.cornell.edu/ mec/2003-2004/cryptography/subs/frequencies.html, 2004.
[18] R. M. Mohammad, F. Thabtah, and L. McCluskey. Predicting phishing websites based on self-structuring neural network. *Neural Computing and Applications*, 25(2), Aug 2014.
[19] R. M. Mohammad, F. A. Thabtah, and L. McCluskey. An assessment of features related to phishing websites using an automated technique. In *7th International Conference for Internet Technology and Secured Transactions*, pages 492–497, 2012.
[20] H. M. Nguyen, E. W. Cooper, and K. Kamei. Borderline oversampling for imbalanced data classification. *Int. J. Knowl. Eng. Soft Data Paradigm.*, 3(1):4–21, Apr. 2011.
[21] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53 – 65, 1987.
[22] E. Sorio, A. Bartoli, and E. Medvet. Detection of hidden fraudulent urls within trusted sites using lexical features. *2013 International Conference on Availability, Reliability and Security*, pages 242–247, 2013.
[23] R. Verma and K. Dyer. On the character of phishing urls: Accurate and robust statistical learning classifiers. In *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, 2015.
[24] C. Whittaker, B. Ryner, and M. Nazif. Large-scale automatic classification of phishing pages. In *NDSS '10*, 2010.
[25] L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *AAAI*, pages 2852–2858. AAAI Press, 2017.

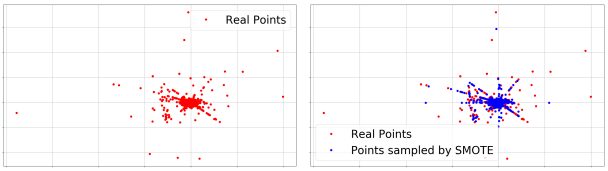
VIII. Appendix

URL Example 3: Phishing URLs we hunted

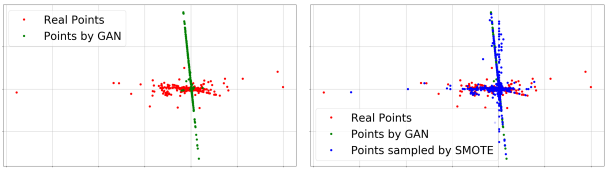
1. http://rainyscape.com/mymatchprofile/
2. http://espaceclientv3−orange−fr.com/dad7c07417f5268c19641b00b7e916ff/
3. http://toplineperformancehorses.com/wp−includes/
4. http://barokahperkasagroup.com/golobababa
5. http://patrickrprince.com/026ea23e4d7e263a24cd3417b3b11be5/
6. http://pousadabangalo.com/rb
7. http://fishingcrowd.com/index/4b3ac8d9884960a9275d4b88eb8b9dd7
8. http://unifiedenergysolutions.com/logos
9. http://pavpal−update−ltd.com/webapps
10. http://casaforsalerealestate.com/gtexx

A. Examples of Phishing URL Hunting

In the above list, we show phishy URL examples we hunted. While these are part of our test sets, we observe that they are exactly generated by the proposed GAN model. One common property of them is that they are relatively shorter than other phishy URLs. We guess this is because the hunting success rate exponentially decreases as the length increases. At the time of submission, many of the generated phishing URLs have already been removed. Recall that phishing attacks do not last long. In almost all cases, they are operated for a short duration and removed from the Internet because they do not want defenders to learn about their attacks.

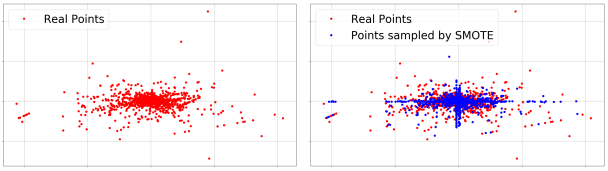


(a) Phishing URLs for Bank of America in our dataset (b) SMOTE oversampling

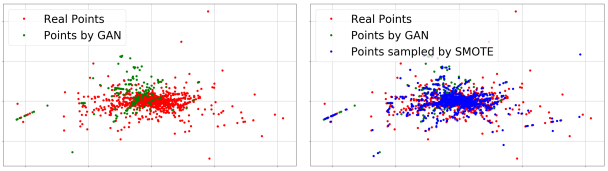


(c) Our proposed text-GAN (d) Our final proposed method

Figure 4: Oversample examples of phishing URLs for Bank of America



(a) Phishing URLs for ebay in our dataset (b) SMOTE oversampling



(c) Our proposed text-GAN (d) Our final proposed method

Figure 5: Oversample examples of phishing URLs for ebay

B. Visualization of Oversampling

Figures 4 and 5 show the oversampling results for Bank of America and ebay. We use the same method that we employed to visualize the PayPal Phishing URLs shown in Figure 3.