Phishing URL Detection: A Network-based Approach Robust to Evasion

Taeri Kim*
Hanyang University
Seoul, Korea
taerik@hanyang.ac.kr

Noseong Park*
Yonsei University
Seoul, Korea
noseong@yonsei.ac.kr

Jiwon Hong
Hanyang University
Seoul, Korea
nowiz@hanyang.ac.kr

Sang-Wook Kim†
Hanyang University
Seoul, Korea
wook@hanyang.ac.kr

ABSTRACT

Many cyberattacks start with disseminating phishing URLs. When clicking these phishing URLs, the victim's private information is leaked to the attacker. There have been proposed several machine learning methods to detect phishing URLs. However, it still remains under-explored to detect phishing URLs with *evasion*, *i.e.,* phishing URLs that pretend to be benign by manipulating patterns. In many cases, the attacker i) reuses prepared phishing web pages because making a completely brand-new set costs non-trivial expenses, ii) prefers hosting companies that do not require private information and are cheaper than others, iii) prefers shared hosting for cost efficiency, and iv) sometimes uses benign domains, IP addresses, and URL string patterns to evade existing detection methods. Inspired by those behavioral characteristics, we present a *network-based inference* method to accurately detect phishing URLs camouflaged with legitimate patterns, *i.e.,* robust to evasion. In the network approach, a phishing URL will be still identified as *phishy* even after evasion unless a majority of its neighbors in the network are evaded at the same time. Our method consistently shows better detection performance throughout various experimental tests than state-of-the-art methods, *e.g.,* F-1 of 0.891 for our method vs. 0.840 for the best feature-based method.

CCS CONCEPTS

• Security and privacy → Phishing; **• Computing methodologies → Classification and regression trees**;

KEYWORDS

phising detection; classification; network-based inference

ACM Reference Format:
Taeri Kim, Noseong Park, Jiwon Hong, and Sang-Wook Kim. 2022. Phishing URL Detection: A Network-based Approach Robust to Evasion. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22), November 7–11, 2022, Los Angeles, CA, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3548606.3560615

1 INTRODUCTION

Cyberattacks cause huge damage to our society. Many cyberattacks start with phishing. Phishing is to trick people into revealing their sensitive information to the attacker. In particular, phishing URLs are camouflaged as URLs that look familiar to people. Careless people will click them, causing their private information to be leaked. Therefore, many detection methods have been developed and as a response, attackers started to consider evasion techniques that camouflage with legitimate patterns (see Section 3 for more details) [18, 29, 38, 39, 45]. Thus, it is of utmost importance to prevent phishing attacks using evasion.

There have been proposed machine learning methods to detect phishing. They can be categorized into two types: content-based and URL string-based. *Content-based methods* download and analyze web page contents [33, 36, 44]. However, they require non-trivial computations to process many web pages and are weak against web browser-based exploits (because we need to access their web pages). Most importantly, it is not easy to collect such training data. For all those reasons, content-based methods are not always preferred. *String-based methods* mainly rely on URL string pattern analyses because it is well known that phishing URLs have very distinguishable string patterns [1–3, 6, 12, 19, 30, 36, 37, 44, 50]. Thus, many lexical features to detect phishing URLs have been proposed (see Section 2). These features are known to be effective in detecting phishing URLs. Because string-based methods are computationally lightweight and provide high accuracy, many researchers prefer them for the high efficiency [44]. Some researchers rely on a blacklist of IP addresses and domains. However, its accuracy is known to be mediocre.

Almost all existing string-based methods hardly consider evasion [45]. Evasion means the technique that the attacker creates phishing URLs seemingly legitimate by manipulating their patterns to deceive defenders' detection methods. In this work, we consider a couple of more key patterns of phishing attacks to design an advanced string-based detection method that outperforms existing methods and is strong against evasion. First, the attacker is sensitive to cost efficiency [15]. In many cases, they (partially) reuse phishing attack materials and prefer specific hosting companies for their looser policies (*e.g.,* not to require identification information) and relatively cheaper prices than other agencies. When a private server is used instead of hosting companies, the attacker prefers shared hosting, *i.e.,* one server is used for multiple phishing attack campaigns and also for multiple domains — in our data, 15.8% of IP addresses are connected to multiple domains. Second, the attacker creates phishing URLs on top of benign servers, domains, IP addresses, and/or substrings to evade existing detection methods [15].

*Two first authors have contributed equally to this work.
†Corresponding author.

CCS '22, November 7–11, 2022, Los Angeles, CA, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9450-5/22/11…$15.00
https://doi.org/10.1145/3548606.3560615

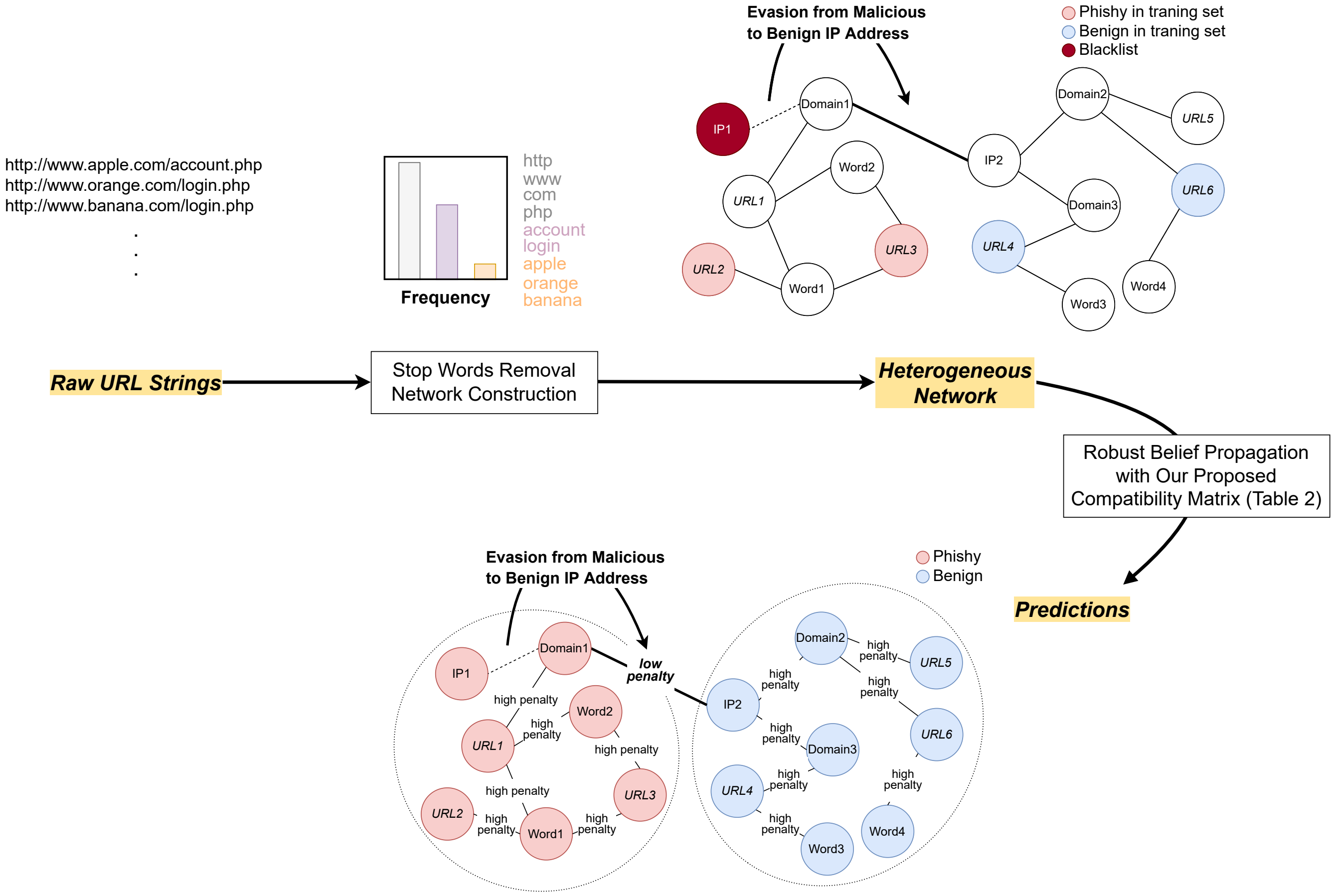


Figure 1: The overall workflow of the proposed method. In the first step, we segment collected URLs into words and remove meaningless ones that correspond to stop words that have high frequency but do not carry useful information. In the second step, we construct a heterogeneous network of URLs, Domains, IP addresses, etc. In the last step, we run the customized belief propagation method to make it robust.

Considering all these facts, we design a novel unified framework of natural language processing and a network-based approach to detect phishing URLs — its overall workflow is shown in Fig. 1. We regard each URL as a sentence and segment it into substrings (words) considering the syntax and punctuation symbols of URLs — URLs have well defined syntax as in English. After that, we build one large network that consists of heterogeneous entities, such as URLs, domains, IP addresses, authoritative name servers, and substrings, and perform our *customized belief propagation* to detect phishing URLs (see Section 4.3.1). We note that the above listed related works do no include any network-based inference schemes. On the contrary, similar network-based inference methods had been used in various other domains [7, 32]. However, our method differs from them in defining *edge potentials* which decide a penalty when two neighboring entities have different predicted labels.

Our approach is effective to infer that seemingly unrelated phishing URLs are actually related and is robust to evasion. Because we infer on such a network of heterogeneous entities, *an evasion for a phishing URL is not likely to be successful unless a majority of its neighbors in the network are evaded at the same time* (see Section 5 for more detailed discussions with theorems and proofs), which is our main contribution in comparison with existing works.

We crawled many suspicious URLs and also downloaded a couple of datasets released by other researchers [11, 46]. In total, we have about 120K phishy and 380K benign URLs. We compare our approach with state-of-the-art baseline methods including graph convolutional networks (GCNs) and feature engineering-based methods. Our method shows the best detection performance among them. Furthermore, in additional evasion tests, our method shows better F-1 scores than other baseline methods. Because the evasion incurs non-trivial expenses for the attacker to access to benign domains, IP addresses, and so forth, our robust detection method greatly increases the attacker's financial burden to perform evasion.

Our contributions can be summarized as follows:

- We design a novel network-based inference method equipped with our proposed robust edge potential assignment mechanism. Our network inference on top of the edge potential assignment outperforms many baseline methods including feature engineering-based and network-based classifiers.

- Our proposed network-based method has a theoretical ground on why it is robust to evasion (see Section 5).
- We conduct experiments with a large set of URLs collected by us and downloaded from other work. Our data covers a wide variety of phishy/benign URL patterns.

In the following, we first review the literature in Section 2 and describe the motivation of this work in Section 3. Then, in Sections 4 and 5, we design a novel network-based detection method robust to evasion and analyze the theoretical robustness of the proposed method. After that, we conduct extensive experiments on phishing URL detection with and without evasion in Section 6. Lastly, in Sections 7 and 8, we describe crawled our data and conclude our paper. For reference, in Appendix A, we introduce a set of lexical features widely used to detect phishing URLs and sorted in descending order of the feature importance extracted from the best performing baseline method.

2 RELATED WORK

In this section, we review phishing URL detection models and attackers' behavioral pattern analyses.

2.1 Methods to Detect Phishing URLs

Extensive work has been done to counter phishing attacks [1, 3, 6, 12, 30, 33, 36, 37, 44, 50]. 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. So, we introduce a widely used set of 19 URL features that we collected from related papers in Appendix A. All these features are used by some baseline methods in our experiments. All the mentioned works are not based on network-based inference but on feature engineering.

Mao et al. designed a phishing URL detection method robust to evasion based on web page content features [33]. However, it is not easy to collect such training data in many cases because phishing attacks do not last long and web pages are quickly removed, which is one common drawback of all content-based detection methods [1].

In [3, 22, 23], several sequence (*e.g.*, URL in our context) classification models have been proposed. Some of them have an advanced architecture to combine various components such as recurrent neural networks, convolutional neural networks, word embeddings, and their multiple hierarchical layers. We use their ideas as additional baselines. The first one uses long short-term memory (LSTM) cells and the second model uses one-dimensional convolution (1DConv), and the third baseline uses both (1DConv+LSTM).

For a couple of related problems [7, 32], network-based methods have been used. In [32], the authors tried to detect malicious domains (rather than URLs) and the authors in [7] proposed one heuristic-based belief propagation method to detect malicious codes. Those two works differ in how to create networks but use the same belief propagation method. Both methods correspond to the baseline method marked as 'POL' in our experiments. Peng et al. and Khalil et al. also tried a network approach for malicious domain detection [21, 40]. However, their methods are not directly applicable to our phishing URL data.

2.2 Attackers' Behavioral Patterns

Phishing Activity Trends Report [15] by Anti-Phishing Working Group is one of the most reputable reports. We analyzed their quarterly reports. The two most important observations from the reports are i) there are some web hosting companies preferred by the attacker due to their low prices and anonymity, and ii) many phishing URLs have similar string patterns because they are created by similar tools or reused from old phishing campaigns. There exist many other interesting observations as follows:

- There has been an increase in the number of phishings using free hosting providers or website builders. It has been reported that 81.7% of malicious websites are hosted on free hosting providers [10]. These free hosts are easy to use but also allow threat actors to create subdomains spoofing a targeted brand, resulting in a more legitimate-looking phishing site. Free hosts also afford phishers additional anonymity, because these services hide registrant information.
- The attacker prefers shared hosting which means multiple domains share the same hosting server. Therefore, seemingly unrelated domains may belong to the same host or IP address.
- Hundreds of vendors are mostly targeted. This continues a years-long trend in which a few hundred companies are attacked regularly. Considering this fact, we crawled URLs from phishtank.com for the three most frequently attacked vendors: Bank of America, eBay, and PayPal.
- 53% of phishing attacks use 'com' domains and 'net', 'org', and 'br' domains are next equally preferred.

3 MOTIVATION

Definition 1. *Evasion is an effective technique that one can adopt to disturb a machine learning task by creating a 'counter-evident' sample, e.g., a phishing URL hosted by a benign domain or IP address. This evasion can be done in various ways. For detailed evasion techniques that we consider, refer to Section 6.6.*

Shirazi et al. showed that existing phishing URL detection methods are adversely impacted by evasion without suggesting a countermeasure [45]. Specifically, they conducted evasion tests that randomly select up to four features of phishing URLs and change the selected features to other benign values. In their non-evasion tests, most classifiers showed high accuracy. In their evasion tests, however, the best performing classifier's accuracy (recall) decreased from 82-97% to 79-45% with one feature change, and to 0% with four feature changes.

To our knowledge, it has not been actively studied to design a non-content-based phishing URL detection method robust to evasion. We consider many aspects of URLs, including domains, IP addresses, name servers, and string patterns *except contents* — because collecting phishing web page contents require non-trivial efforts. Most importantly, our method is based on a network of them. Intuitively speaking, attackers cannot disturb our network-based inference task even after evasion if many neighbors of a phishing URL in the network remain the same as before (see Section 5). Some large-scale evasion can still neutralize our method. However, it requires non-trivial expenses, thus decreasing the attackers' motivation on such evasion.

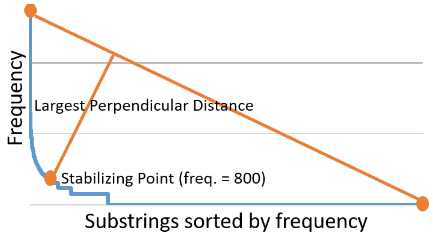


Figure 2: The elbow method to find the stabilizing point of frequency. All substrings (words) before the found stabilizing point are considered as stop words.

While it is hard to measure the evasion cost for money, it includes various intangible efforts, such as exploiting benign web servers to implant their phishing pages, maintaining a custom domain without any phishing campaigns until D-Day to prevent it from being blacklisted, and so forth. In particular, it depends on security environments and skills how long it will take until an attacker successfully exploits an administrator's account of a benign server.

4 PROPOSED METHOD

After introducing the overall workflow in our method, we describe its detailed steps with some key visualization results.

4.1 Overall Method

Fig. 1 shows our overall workflow. The entire process can be divided into the following steps:

(1) We crawl many URLs from phishtank.com and download other works' open datasets.
(2) As mentioned earlier, we create a heterogeneous network of URLs, domains, IP addresses, name servers, and substrings (words). We use a standard natural language processing technique to segment URLs into substrings (words) and draw edges between a URL and substrings.
(3) We run our customized belief propagation algorithm to infer unknown URLs' phishy/benign labels, which is our main contribution. In particular, this type of inference is called *transductive*. In our case, both training and testing samples co-exist in a network and testing samples' labels are inferred from other known training samples' labels following the network architecture.

4.2 Network Construction

We do a network-based classification rather than feature engineering-based classification. As mentioned earlier, phishing URLs share many common string patterns and various entities are cross-related to each other, so we create a network to represent complicated relationships among multiple entities (vertices) such as URLs, their domains, IP addresses, authoritative name servers, and substrings.

- We draw an edge between a URL and its domain.
- We draw an edge between a domain and its resolved IP address. We use domains.google and virustotal.com to retrieve domain-IP address resolution history. They return not only current but also all past resolution results with timestamps which enable correct connections. Sometimes, one domain can be connected to multiple IP addresses.

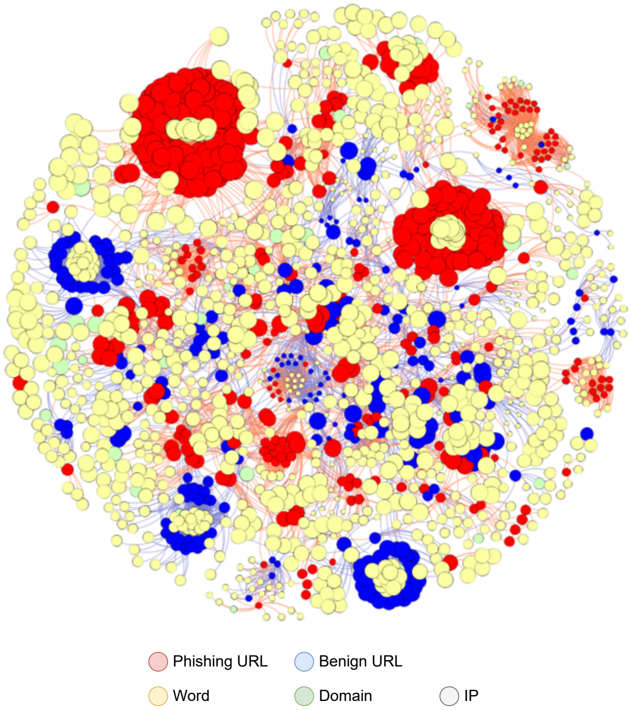


Figure 3: The network constructed from our data. Red means phishing URLs and blue means benign URLs. Other colors mean non-URL entities — name servers are not displayed due to their lesser significance than that of other entities. Note that there exist many clusters. The vertex size represents the strength (more specifically, modularity [5]) of the cluster that a vertex belongs to.

- We draw an edge between a domain and its authoritative name servers. In general, there exist multiple authoritative name servers for a domain, and one authoritative name server provides resolution services for multiple domains.
- We draw an edge between a URL (*i.e.,* sentence) and a substring (*i.e.,* word) if the URL contains the substring. For these edges, it is very crucial how to segment a URL into substrings. We will shortly describe this in the following section.

4.2.1 How to segment a URL into words. A URL is used to locate resources in the Internet. It consists of several parts: scheme, username, password, host, port number, path, and query string — some of them can be missing. We use our customized word segmentation policies in each part as follows:

- *Scheme* means the protocol, *e.g.,* http and https. Only two words can be possible. However, since these words have very high frequencies, we do not use these two words in our network. We will describe how to remove those *stop words*[1] of URLs shortly.

[1] Stop words do not carry meaning but have high frequency values in English, such as 'a', 'the', 'is', and so forth. It is a standard process to remove such stop words in natural language processing algorithms. We use the elbow method based on frequency to detect the stop words of URLs.

- *Username* and *password* can be specified before host. We segment them using the punctuation symbols, *i.e.,* ‘//’, ‘:’, and ‘@’. An example is ‘http://username:password@example.com’.
- *Hostname* can be simply segmented into words by ‘.’.
- Sometimes *path* can be very long, separated by ‘/’. We use all possible punctuation symbols, such as ‘/’, ‘.’, ‘!’, ‘&’, ‘,’, ‘#’, ‘$’, ‘%’, and ‘;’, to segment the path part into words.
- *Query string* is able to contain multiple queries separated by ‘&’, and each query consists of a query name and a value, *e.g.,* ‘term=bluebird&source=browser-search’. We extract words using the two punctuation symbols, ‘=’ and ‘&’.

Because the syntax of URLs is well defined, extracting words can be done very efficiently. However, many meaningless words can be also extracted. Therefore, before drawing edges between URLs and words, those words should be removed. In the field of natural language processing, it is well known that the frequency of words follows Zipf's law — more precisely, word frequency exponentially decays [47]. In particular, this pattern describes stop words in English very well. For instance, the frequency of the most popular stop word ‘the’ occupies 7% of all word occurrences in the Brown Corpus of American English [13] and the second most popular stop word ‘of’ has 3.5%. We found that the extracted words from URLs show similar statistics (cf. Fig. 2). After that, we remove some high-frequency words using the *elbow method* [20]. It decides the point whose perpendicular distance to the line segment connecting the two ends is the biggest as the saturation point, which is 800 in our data. We remove all the words whose frequency values are larger than the point.

Fig. 3 shows the network created by the proposed method. Note that there exists strong correlation between the cluster constructions and the ground-truth phishy/benign labels, which justifies our network-based inference method that will be described shortly. *In this regard, the main intuition in our work is that it is hard to evade our natural language processing and network-based approach unless a majority of entities in a cluster are evaded simultaneously.*

4.3 Network-based Inference

We employ *loopy belief propagation* (LBP) [4] for our network-based inference. Our key contribution in this step is to define a more advanced edge potential assignment mechanism than that of the state-of-the-art methods [7, 32]. Because these methods typically not only follow a majority voting of neighbors but also give a fixed edge potential regardless of the similarity of the two connected vertices, a vertex is mainly classified as benign if it has many benign neighbors. However, we want to correctly classify a phishing vertex even if it has many benign neighbors. Therefore, we define a more advanced edge potential assignment mechanism for enabling more sophisticated classification and achieving evasion-robustness. We will describe our edge potential definition in Section 4.3.1.

LBP is a message passing algorithm to solve network-based inference problems. Let $x \in X$ be a hidden variable and N_x be a set of its neighboring variables, and let $o \in O$ be an observed variable. In our contexts, an observed variable means a training sample and a hidden variable means a testing sample. We use X and O to denote a set of all hidden and observed variables, respectively. Each variable represents the phishy/benign label of an entity in our case. x sends a message to other hidden variable $y \in N_x$ after collecting all messages from $N_x \setminus \{y\}$. Note that observed variables never receive any messages; they only broadcast the messages to their neighboring hidden variables. In our case, phishy and benign URLs in the training set are observed variables.

As mentioned, we need to calculate a message $msg_{x \rightarrow y}(\ell)$ from a variable x to other variable y regarding a phishy/benign label $\ell \in L$, where $L = \{phishy, benign\}$ is a set of all possible label options. There exist several message passing strategies: *sum-product*, *max-product*, and *min-sum*. We use the min-sum algorithm having better computational stability than the other two algorithms. For some high degree vertices, message values tend to quickly decay to zeros (*i.e.,* floating point underflow) in the sum-product and max-product. Their product operation is reduced to the sum in the min-sum algorithm. The message in the min-sum algorithm is calculated as:

$$\begin{aligned} msg_{x \rightarrow y}(\ell) = \min_{\ell'} \Big[\log\left(1 - \phi_y(\ell')\right) + \psi_{xy}(\ell, \ell') + \\ \sum_{k \in N_x \setminus \{y\}} msg_{k \rightarrow x}(\ell') \Big], \end{aligned} \tag{1}$$

where $\phi_y(\ell')$ is a *prior* that the variable y has the label ℓ' and $\psi_{xy}(\ell, \ell')$ is an *edge potential*, indicating a joint-probability that x's label is ℓ and y's label is ℓ'. Note that there is a log function in the message definition so the min-sum is equivalent to performing the max-product in the log space for better computational stability.

After exchanging messages many times, we first calculate a *cost* of each variable and label pair and then choose the label that yields the *lowest cost*[2] for each variable. The cost, when x has the label ℓ, is computed as:

$$Cost(x, \ell) = \log\left(1 - \phi_x(\ell)\right) + \sum_{k \in N_x} msg_{k \rightarrow x}(\ell). \tag{2}$$

Then, the formal definition of the problem that the min-sum algorithm solves can be defined as follows:

$$\text{argmin}_g \sum_x Cost(x, g(x)), \tag{3}$$

where $g : X \rightarrow L$, where X is a set of hidden variables and $L = \{phishy, benign\}$, is a label assignment function. It is worth mentioning in our setting, x can be a hidden variable representing a URL, domain, IP, name server, or word. Our final target is to infer the labels of testing URLs. To this end, we need to infer the labels of other non-URL entities as well because they connect URLs. Therefore, the min-sum algorithm can be described as a process of finding such label assignments to hidden variables that the sum of the costs is minimized.

4.3.1 Edge Potential Assignment. The definition of edge potential $\psi_{xy}(\ell, \ell')$ is the key factor in the LBP method. [7] used the heuristics of *homophily* and *heterophily*. They, for example, assign an edge potential of $0.5 - \epsilon$ (resp. $0.5 + \epsilon$) if two neighboring variables x and y have different (resp. same) labels as shown in the compatibility matrix in Table 1. ϵ is usually set as very small, *e.g.,* 0.001. We use two labels, phishy and benign. At the end of the network-based

[2] The min-sum tries to minimize ‘cost’ as the name ‘min’ suggests whereas both the sum-product and max-product maximize ‘belief’.

Table 1: The compatibility matrix proposed in Polonium [7] based on the homophily heuristic

$\psi_{xy}(\ell, \ell')$	Phishy	Benign
Phishy	$0.5 + \epsilon$	$0.5 - \epsilon$
Benign	$0.5 - \epsilon$	$0.5 + \epsilon$

Table 2: Our compatibility matrix M for the min-sum algorithm. x and y mean vector representations. $sim(\mathbf{x}, \mathbf{y})$ is a similarity between two vectors.

$\psi_{xy}(\ell, \ell')$	Phishy	Benign
Phishy	$\min(ths_+, 1 - sim(\mathbf{x}, \mathbf{y}))$	$\max(ths_-, sim(\mathbf{x}, \mathbf{y}))$
Benign	$\max(ths_-, sim(\mathbf{x}, \mathbf{y}))$	$\min(ths_+, 1 - sim(\mathbf{x}, \mathbf{y}))$

inference process, for each entity, one label will be assigned as a prediction result. The final label assignments are greatly influenced by the edge potential definition.

In contrast to [7], we incorporate more factors, such as similarity among entities and an improved compatibility matrix, to derive reliable edge potentials — we prove shortly in Section 5 that reliable similarity definitions can lead to the evasion-robustness in our method. The similarity can be measured via various embedding approaches, such as Doc2Vec [24] and Node2Vec [16]. We discuss how to calculate vector representations of URLs, their domains, IP addresses, authoritative name servers, and words in Section 4.3.2.

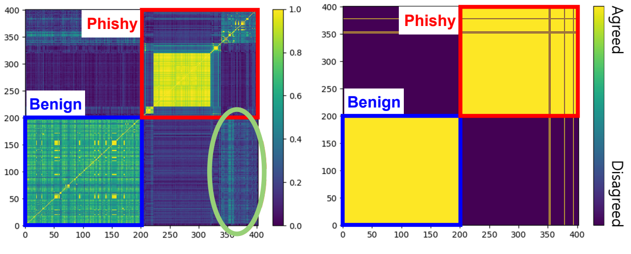
To calculate the similarity based on those vector representations, we adopt several different similarity measures, including the cosine similarity and various kernels. Our proposed definition of edge potential is shown in Table 2. In the table, we denote vector representations of entities in boldface and $sim(\mathbf{x}, \mathbf{y})$ indicates a similarity between two vectors that can be defined in various ways. Two such examples are as follows:

$$sim(\mathbf{x}, \mathbf{y}) = \begin{cases} cos(\mathbf{x}, \mathbf{y}) \text{ based on the cosine similarity,} \\ \exp(\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}) \text{ based on the RBF kernel.} \end{cases}$$

After that, we use a concept inspired by the *hinge-loss* [42] to assign edge potential values. For instance, $\min(ths_+, 1 - sim(\mathbf{x}, \mathbf{y}))$ in the table is to limit the minimum edge potential to ths_+[3] when two entities have the same label. When $sim(\mathbf{x}, \mathbf{y})$ is low (resp. high), the proposed definition imposes a large (resp. small) penalty closed to 1 (resp. ths_+). Therefore, the proposed mechanism is able to assign much more sophisticated edge potentials in comparison with existing methods.

One should be very careful when applying our compatibility matrix to other applications. Recall that we use the min-sum algorithm so that in our compatibility matrix M, we assign 0 (which corresponds to 1 in the sum-product and max-product algorithms) when ℓ and ℓ' are the same. For the sum-product and max-product algorithms, $1 - M$ should be used.

4.3.2 Vector Representations of Entities. We describe how we can calculate reliable vector representations of various entities. These embedding methods are known to be effective in discovering latent relationships among entities [16, 24–27, 35, 41, 52], which is a good fit to our network-based detection under the presence of evasions.



(a) Cosine Similarity

(b) Inference Result

Figure 4: Examples of the pairwise vertex similarity with DeepWalk and our network inference with $ths_+ = ths_- = 0.7$. (a) We choose the highest PageRank URL and other 199 URLs in its neighborhood with the breadth-first search for each of the phishy and benign classes. In total, there are 400 URLs in the similarity plot. (b) From the similarity, our network-based inference is able to infer almost correctly.

Word Embedding-based Methods. In the area of natural language processing, there have been proposed various semantic embedding methods such as Word2Vec [35] and Doc2Vec [24]. As we mentioned earlier, we segment URLs into words so we can directly apply the methods to calculate the vector representations of URLs and words. However, we cannot directly calculate vector representations of domains, IP addresses, and name servers in this approach because it considers only strings. Inspired by *locally linear embedding* (LLE) [43], however, we propose a heuristic to represent a domain, IP address, or name server as a mean vector of its neighbors' vectors. LLE says that a vector representation of an entity is a weighted combination of its neighbors' vectors, *e.g.*, equally weighted in our case. For this, we first calculate mean vector representations of domains and then IP addresses and so forth, given URLs' vector representations calculated by Word2Vec or Doc2Vec.

Network Embedding-based Methods. Another reliable approach to find vector representations is to use network embedding methods. Many of these methods have been proposed by social network researchers. One advantage of the approach is that we can find vector representations of all entities simultaneously because they can run on our network directly. We use Node2Vec [16] and DeepWalk [41]. In Fig. 4 (a), we show a pairwise similarity plot that intuitively justifies our embedding and similarity-based edge potential assignments. However, we see a small portion of phishy and benign pairs in the green circle have high similarities. This can be corrected by our proposed edge potential assignment mechanism, which is shown in Fig. 4 (b).

5 EVASION-ROBUSTNESS OF OUR NETWORK-BASED APPROACH

In this section, we formally prove that a hidden variable's phishy/benign label follows its *similar* neighbors' majority label, which improves the robustness to evasion.

Lemma 1. *Suppose $ths_+ = ths_- = 0$ and a small network that consists of a hidden variable u and its m neighbors $N_u = \{v_1, \cdots, v_m\}$.*

[3] This means a lower-bound of edge potential and is set by a user.

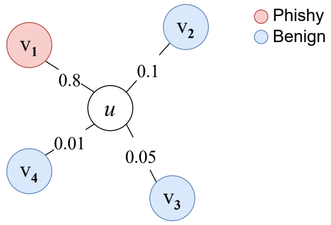


Figure 5: For ease of discussion, suppose u is a hidden variable and other variables' labels are fixed. Each edge is annotated with $sim(\mathbf{u},\mathbf{v}_i)$. Our method concludes that u is phishy although u has more benign neighbors.

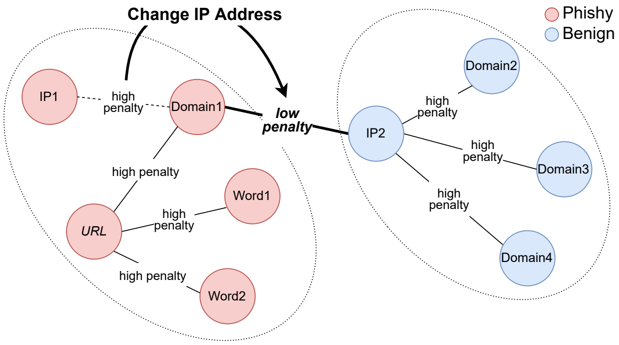


Figure 6: There are two clusters. In general, connections between phishy and benign clusters are not strong (cf. Fig. 3). 'Domain1' is connected to 'IP2' after evasion. However, the connection between them is weak and after embedding, $sim(\mathbf{x},\mathbf{y})$ is low, where $x = Domain1$ and $y = IP2$. Thus, a low penalty is given to their dissimilar labels by our compatibility matrix and the belief propagation can still identify 'Domain1' as phishy.

Let ℓ_u be the phishy/benign label of u. When $\ell_u = \text{argmin}_\ell \sum_j sim(\mathbf{u},\mathbf{v}_j) \cdot I(v_j,\ell)$, where $I(v_j,\ell) \in \{0,1\}$ is an indicator function saying if v_i has a label ℓ, the min-sum algorithm in Eq. (3) *is optimized.*

Proof. ℓ_u is inferred by Eq. (2). In particular, the second term in the equation, $\sum_{v \in N_u} msg_{v \to u}(\ell)$, is significant to decide its label, and $msg_{v \to u}(\ell)$ is dominated only by $\psi_{vu}(\ell_v,\ell_u)$ in the assumed network (cf. Eq. (1)). $\sum_{v \in N_u} \psi_{vu}(\ell_v,\ell_u)$ is minimized when ℓ_u follows the majority label considering the vector similarities because $\psi_{vu}(\ell_v,\ell_u)$ is determined by $sim(\mathbf{v},\mathbf{u})$ in Table 2. □

Example 1 (Example of Lemma 1). *In Fig. 5, there is the small network we used in Lemma 1. For ease of discussion, suppose that only u is a hidden variable and others are observed variables. The optimal min-sum solution is $g(u) = Phishy$ because $sim(\mathbf{u},\mathbf{v}_1) > \sum_{j>1} sim(\mathbf{u},\mathbf{v}_j)$ and $Cost(u,Phishy) = \sum_{j>1} sim(\mathbf{u},\mathbf{v}_j)$ is smaller than $Cost(u,Benign) = sim(\mathbf{u},\mathbf{v}_1)$.*

This lemma can be generalized to the following theorem for larger general networks:

Table 3: The number of phishy and benign URLs for each dataset. Note that Sorio's and Ahmad's datasets are already tagged with ground-truth labels, so we did not use virustotal.com for them. There exist overlapped URLs so the total number of URLs is smaller than their sum.

Dataset	VirusTotal Threshold	# Phishing URL	# Benign URL
Bank of America	4/7	4,610	9,408
eBay	4/7	8,529	18,800
PayPal	4/7	9,690	17,572
Sorio et al. [46]	N/A	40,439	3,637
Ahmad et al. [11]	N/A	62,231	344,800
Total	N/A	119,012	381,734

Theorem 1. *Given a large network $G = (V,E)$, the min-sum algorithm is optimized if for each hidden variable $u \in V$ and its neighbors N_u, $\ell_u = \text{argmin}_\ell \sum_j sim(\mathbf{u},\mathbf{v}_j) \cdot I(v_j,\ell)$.*

Proof. If we can achieve $\ell_u = \text{argmin}_\ell \sum_j sim(\mathbf{u},\mathbf{v}_j) \cdot I(v_j,\ell)$ for each hidden variable u, it is immediate that the overall cost is minimized in Eq. (3) because the overall cost is defined as the sum of each hidden variable's cost. □

This theorem discusses a sufficient condition of the optimal min-sum solution but sometimes the sufficient condition, $\ell_u = \text{argmin}_\ell \sum_j sim(\mathbf{u},\mathbf{v}_j) \cdot I(v_j,\ell)$ for each $u \in X$, is not achievable. However, what the min-sum does in such a case is to strategically drop the sufficient condition for some hidden variables to better minimize the sum of costs for other majority of hidden variables. Therefore, we can still say that the sufficient condition is achievable in general in any network for its majority of hidden variables. In particular, our embedding and hinge-loss based edge potential assignment bring large flexibility to the process. Therefore, the cost sum can be effectively minimized with the proposed method. Fig. 4 shows one such example that our proposed method is able to achieve the sufficient condition in most cases by ignoring some minor edges with high similarity. Because of this property, our approach is robust to evasion unless the attacker *collectively evade* for neighboring URLs/domains/IP addresses/name servers (see Fig. 6 for an example). However, the collective evasion will cost non-trivial expenses to the attacker.

6 EXPERIMENTS

In this section, we introduce our detailed experimental environments and results. We collected many URLs from crowd-sourced repositories and other papers. After that, we conducted experiments with ten baselines, ranging from classical classifiers and graphical methods to graph convolutional networks. Our method shows the best accuracy and robustness.

The source codes, data, and reproducibility information of our method are available at https://github.com/taerikkk/BPE.

6.1 Datasets

There have been created several phishing URL detection datasets [30, 37, 51]. However, almost all of them do not release raw URL strings so we cannot use their datasets. We found only two open datasets with raw URL strings [11, 46]. In addition to them, we also crawled phishtank.com and collected three sets of URLs reported during a couple of months recently for Bank of America, eBay, and PayPal, the top-3 most popular targets in the website (see Section 7 for more details). Phishtank.com is a crowdsourced repository of suspicious URLs that does not provide ground-truth labels — users can upvote or downvote the reported URLs in the website, but its voting system is not reliable because anyone (even including attackers) can participate. In total, we have about 500K URLs, 172K domains, and 66K IP addresses. 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 seven most reliable and popular AV products (such as McAfee, Norton, Kaspersky, Avast, and Trend Micro) were selected among them, and a URL is considered phishy if more than half of them indicate so, *i.e.,* tagging by majority vote. At the end, we merged these datasets into one and created a very large URL dataset whose statistics are shown in Table 3.

We split the combined set in the standard ratio of 80:20 for training and testing. Only 10% of the URLs have timestamps. With them, we also tried to split in chronological order. Our method shows good accuracy for this configuration as well. However, we do not include the results because of i) its results similar to that of the random split, ii) its small data size, and iii) space reasons.

6.2 Baselines and Hyperparameters

Among many methods proposed, we consider the following baseline methods in our experiments. First, we test many feature-based prediction models. For this, we had surveyed literature and collected 19 features (see Appendix A). After that, we predict with various classifiers after under/oversampling to address the imbalanced nature in our dataset — benign URL numbers are much larger than phishing URL numbers in the training set. In addition to synthetic minority oversampling [8] and adaptive synthetic sampling [17], we consider five undersampling methods, six oversampling methods, and one ensemble method below.

Undersampling methods:

- Naive random undersampling is randomly choose samples to drop.
- Tomek's link is a representative undersampling method.
- Clustering uses centroids of clusters after dropping other cluster members.
- NearMiss is also popular for undersampling.
- Various nearest neighbor methods are able to undersampling.

Oversampling methods:

- Naive random oversampling is randomly choose samples to add.
- SMOTE [8] and its variants are a family of the most popular oversampling methods, which include five variations.
- ADASYN [17] is also popular for oversampling.

Ensemble method:

- Ensemble method means that we use both the oversampling and undersampling methods at the same time.

We refer to a survey paper [28] for more detailed information. The combination of classifiers, under/oversampling methods, and their hyperparameters create a huge number of possible options in this method. So, first, we perform 5-fold cross validation to choose the best performing classifier/sampling method, and its hyperparameters. Second, we also test three deep learning-based sequence classification methods mentioned in Section 2. These neural networks are based on recurrent or convolutional layers. We use their hyperparameters recommended in their original publications.

Third, on the *simple network* that consists only of URLs and their words, we run the following graphical methods: i) Random Walk with Restart (RWR): This method runs many random walks from training URLs and counts the number of visits to each testing URL. It is very successful for recommender systems [9]. ii) Polonium (POL): Polonium based on a simple belief propagation strategy showed a big success in predicting malware and malicious domains. We run the belief propagation on our network with Polonium's compatibility matrix definition in Table 1. iii) Belief Propagation with Enhancements (BPE): This is our method to run the belief propagation based on our improved definition of compatibility matrix. We test various embedding techniques, $ths_+ = \{0, 0.1, 0.3, 0.5, 0.7, 0.9, 1\}$, $ths_- = \{0, 0.1, 0.3, 0.5, 0.7, 0.9, 1\}$, and for calculating the vector similarity, the cosine similarity and RBF kernel. We set the dimension of the embeddings to 128.

Fourth, on the *extended network* that consists of all entities (cf. Section 4.2), we test the same set of graphical methods: RWR, POL, and BPE. For this, we use the blacklist of 41,881 IP addresses and 158,271 domains provided by virustotal.com. In other words, those blacklisted entities are converted into observed variables and excluded from the inference process. We also test BPE on the *noisy network* where stop words are not removed. Last, we test state-of-the-art graph convolutional networks (GCNs) such as LGCN [14] and GAT [49] on the extended network. For a vertex v, we feed a feature vector after concatenating i) the 19 features of v we use in the feature-based prediction, ii) a binary value denoting whether v is blacklisted or not, and iii) a one-hot vector where only the index of the vertex v is one. If some items are missing, we concatenate with zeros — *e.g.,* a domain does not have the 19 features so we zero them out. We test the hyperparameters recommended in their original papers. To prevent overfitting, we also add an L2 regularization of neural network weights. In all those graphical models, such as RWR, POL, GCNs, and BPE, the labels of training URLs are fixed and only unknown labels of testing URLs are inferred.

We exclude other content-based detection methods in our experiments because it is hard to obtain web page contents in general — recall that phishing attacks do not last long and attackers usually clean their traits from the Internet after the accomplishment of their goal. The two datasets we downloaded from [11, 46] do not include any content information and we also could not collect web page information in HTML from phishtank.com in a stable manner.

6.3 Environments

Hardware. We conducted our experiments on the machines with i9-9900K, 64GB RAM, and GTX 1070.

Table 4: Detection results of some selected baseline methods and our proposed method. The best result in each measure (*i.e.,* each column) is indicated in boldface.

Type	Method	Recall (Phishy)	Precision (Phishy)	F-1	Accuracy
Baseline	AdaBoost	0.830	0.830	0.830	0.831
	SGDClassifier	0.762	0.720	0.734	0.720
	RandomForest	0.840	0.850	0.840	0.847
	LSTM	0.697	0.710	0.688	0.857
	1DConv	0.677	0.735	0.689	0.864
	1DConv+LSTM	0.788	0.806	0.784	0.902
Noisy Network	BPE	1.000	0.001	0.001	0.083
Simple Network	RWR	0.569	0.917	0.702	0.815
	POL	0.901	0.853	0.876	0.943
	BPE (Cos, Deepwalk)	0.901	0.864	0.882	0.945
	BPE (RBF, Doc2Vec)	0.895	0.864	0.879	0.943
Extended Network	RWR	0.648	**0.930**	0.764	0.863
	POL	0.899	0.850	0.874	0.942
	LGCN	**0.999**	0.762	0.865	0.762
	GAT	0.995	0.762	0.863	0.760
	BPE (Cos, Deepwalk)	0.958	0.831	0.890	**0.969**
	BPE (RBF, Deepwalk)	0.958	0.832	**0.891**	**0.969**

Software. As our experiments utilize many different types of baseline methods, our software environments are rather complicated. The selected list of important software/libraries are as follows:

- Python ver 3.8.1.
- Scikit Learn ver 0.22.1.
- TensorFlow ver 1.5.1.
- CUDA ver 10.
- NetworkX ver 2.4.

6.4 Experimental Results

We summarize the results shown in Table 4 as follows. Among all feature-based methods, RandomForest performs the best. For all metrics, it outperforms AdaBoost, SGDClassifier, and others, *e.g.,* the F-1 score of 0.840 for RandomForest vs. 0.830 for AdaBoost vs. 0.734 for SGDClassifier. However, all these feature-based baseline methods are clearly beaten by the network-based methods. This supports the efficacy of our network-based approach.

RWR's precision for the phishy class on the extended network is the best (0.930). However, its recall is worse than other network-based inference methods. POL shows a balanced performance between recall and precision as in its original task to detect malware. LGCN's recall for the phishy class is the best (0.999). For LGCN and GAT, we found that they are sensitive to hyperparameters and hard to regularize the overfitting. Surprisingly, the best F-1 was made when we allow overfitting to the phishy class to some degree. When we increase the coefficient of the L2 regularizer to prevent overfitting, their F-1 scores drastically decrease. We also found that training with subgraphs is not effective in processing our large network. Therefore, we set the size of subgraph as large as possible in our recent GPU model — due to the GPU memory limitation, whole graph training is impossible for our network — but its performance is inferior to our method.

Our method with $ths_+ = 0.7$, $ths_- = 0.7$, RBF kernel, and DeepWalk, which is marked as 'BPE (RBF, Deepwalk)', shows the best performance for F-1 and accuracy. Although BPE's precision for the phishy class is a little lower (0.832) than the best feature-based method (*i.e.,* RandomForest)'s score (0.850), the BPE's recall for the phishy class is much higher (0.958) than that of RandomForest (0.840). However, one may be worried that our method mis-classifies benign as phishy due to its relatively low precision. To this end, we measure the false positive rate (*i.e.,* FPR) to BPE and RandomForest. As a result, we could obtain 0.031 for BPE and 0.306 for RandomForest. Therefore, we expect that BPE is the most useful for accurately detecting phishing URLs in practice.

The same network-based method on the noisy network shows poor performance (*e.g.,* 0.01 for F-1), which proves our network definition also plays an important role.

Statistical significance. For the statistical significance of our experiments, we conduct paired t-tests with a 95% confidence level between BPE and each baseline, and achieved a p-value less than 0.05 for all cases.

Table 5: F-1 scores of BPE, POL, and RandomForest (RF) after M1-5 evasions. The best result in each evasion method and ratio is indicated in boldface.

Evasion Ratio	M1 (domain)			M2 (path)			M3 (query)			M4 (domain and path)			M5 (domain and query)		
	BPE	POL	RF	BPE	POL	RF	BPE	POL	RF	BPE	POL	RF	BPE	POL	RF
5%	**0.866**	0.836	0.812	**0.876**	0.843	0.820	**0.888**	0.867	0.821	**0.861**	0.811	0.813	**0.873**	0.822	0.814
10%	**0.847**	0.817	0.816	**0.861**	0.822	0.810	**0.882**	0.841	0.816	**0.829**	0.778	0.804	**0.863**	0.811	0.807
15%	**0.833**	0.802	0.810	**0.858**	0.817	0.803	**0.882**	0.836	0.811	**0.805**	0.760	0.790	**0.854**	0.807	0.798

Table 6: F-1 scores of BPE, POL, and RandomForest (RF) after M6 and M7 evasions. The best result in each evasion method and ratio is indicated in boldface.

Evasion Ratio	M6 (path and query)			M7 (all)		
	BPE	POL	RF	BPE	POL	RF
5%	**0.874**	0.838	0.814	**0.861**	0.827	0.760
10%	**0.869**	0.832	0.808	**0.828**	0.791	0.751
15%	**0.857**	0.820	0.802	**0.803**	0.762	0.733

Transductive vs. Inductive. Transductive and inductive inferences are two popular paradigms of machine learning [48]. Among all the baseline methods, RandomForest and some other classifiers are inductive methods and many other network-based methods are transductive. In many cases, people rely on the inductive inference where a generalized prediction model trained with a training set predicts for unknown testing samples. In our work, however, we adopt a transductive method where the class label of a specific unknown testing sample is inferred from other specific related training samples in the network architecture. Fig. 3 justifies our transductive approach because a cluster usually consists of vertices from the same class. However, it is not the case that all transductive methods are successful in Table 4.

Time performance. BPE is an advanced LBP-based method with our novel similarity-based edge potential assignments. Therefore, the time complexity of BPE is $O(S \cdot |E| \cdot t)$, where S indicates a similarity calculation cost, E indicates the set of edges, and t does the number of iterations required for the convergence. t is typically small in our setting, *e.g.,* $t = 5$ is enough. The time complexity of RandomForest (*i.e.,* the best feature-based method) is $O(f \cdot n \cdot \log(n))$, where f is the number of features and n is the number of URLs. In our experiments, the training (wall-clock) time of BPE is 4.7 times faster than that of RandomForest.

6.5 Parameter Sensitivity

Sensitivity to thresholds. The following threshold combinations perform very well and are comparable to each other in our experiments: ($ths_+ = 0.7$, $ths_- = 0.7$), ($ths_+ = 0.3$, $ths_- = 0.9$), ($ths_+ = 0.3$, $ths_- = 0.5$), ($ths_+ = 0.7$, $ths_- = 0.9$), ($ths_+ = 0.5$, $ths_- = 0.3$), and so on. One common characteristic of them is that two extreme values, 0 and 1, are not preferred. This supports our decision to adopt thresholds because two dissimilar neighbors do not always mean that their labels should be different. In other words, the one you are not close to is not necessarily your enemy. By limiting the penalty, we achieved the best accuracy in our experiments.

Sensitivity to embedding. It turns out that network embeddings are more effective than word or document embedding methods. All high ranked results are produced by DeepWalk. Doc2Vec produces the best result only for the simple network and RBF kernel environment. We think that this is because our network definition considers common words among URLs and DeepWalk is able to capture the semantic meanings of words closely located in the network.

Cosine similarity vs. RBF kernel. It seems the cosine similarity and the RBF kernel are comparable to each other in our experiments. When sorting all results, all highly ranked results are evenly distributed to both of them.

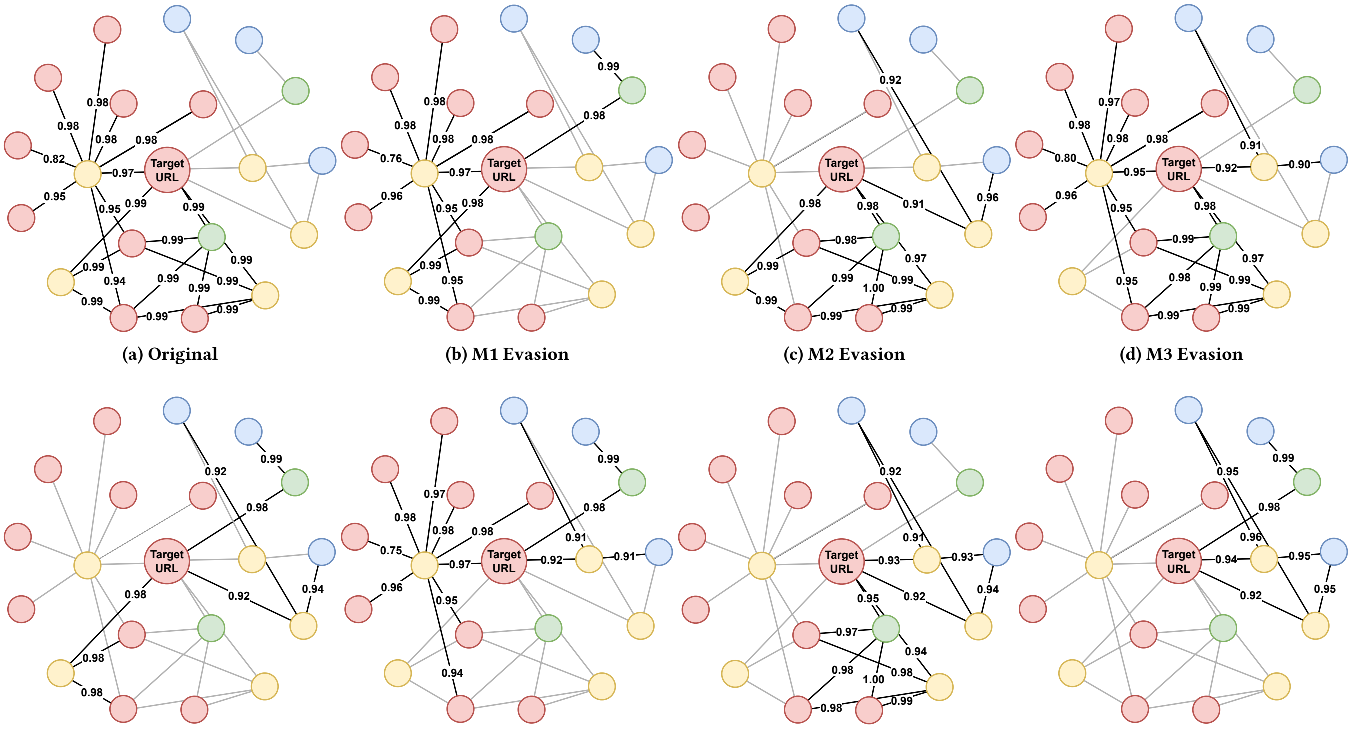
6.6 Evasion Tests

For our evasion testing, we consider all possible variations for the parts of phishing URLs, *i.e.,* domain, path, and query. Specifically, we define, in total, seven evasion methods (*i.e.,* M1-7) as follows: M1) Phishing URL's domain is changed to other random benign domain (and as a result, IP address is changed too); M2) Phishing URL's path string (cf. Section 4.2.1) is changed to other random benign one; M3) Phishing URL's query string (cf. Section 4.2.1) is changed to other random benign one; M4) Phishing URL's domain and path string are changed to other random benign ones; M5) Phishing URL's domain and query string are changed to other random benign ones; M6) Phishing URL's path and query strings are changed to other random benign ones; M7) Phishing URL's each part is independently changed to other random benign ones, *i.e.,* phishing URL becomes an entirely new URL that looks benign.

Note that our evasion tests embrace Shirazi et al.'s evasion settings (cf. Section 3). Also, we note that M7 evasion is the most challenging situation. As mentioned earlier, for M7 evasion, the attackers' motivation may be low, because it requires non-trivial expenses. Nevertheless, it is worth mentioning that we take into account the case where all of the domain, path, and query strings are evaded simultaneously.

For some spear phishing attacks aiming at particular targets, however, sophisticated URLs are prepared with all benign string patterns and web page contents, in which case more advanced techniques are required to detect. It is well known that the attacker invests large efforts in spear phishing by considering even the psychological and habitual characteristics of the targets after hijacking benign user accounts [18, 29, 39]. However, it is out of the scope of this paper and we leave it as our future work.

To simulate evasions, we modify random 5-15% of our testing phishing URLs in each evasion method. After the modifications, its network is also reconstructed accordingly. We compare BPE



(a) Original (b) M1 Evasion (c) M2 Evasion (d) M3 Evasion

(e) M4 Evasion (f) M5 Evasion (g) M6 Evasion (h) M7 Evasion

Figure 7: Visualization of the original network and M1-7 evasions of the target phishing URL denoted with the largest red vertex. Each edge is annotated with the similarity. The meaning of vertex color follows that in Fig. 3.

with POL and RandomForest (RF) which represent network-based and best feature-based baseline methods, respectively. Because all entities are connected in our network and one evasion may affect other neighboring non-evaded URLs in the worst case, a simple measure counting the number of successful detections for the phishing URLs with evasion is not a correct metric. So, we re-evaluate all testing URLs again after evasion and report the results in Tables 5 and 6.

As shown in Tables 5 and 6, BPE outperforms other baselines with non-trivial margins. Especially, BPE outperforms RandomForest by up to 13.29% in the most challenging situation, *i.e.,* M7 with an evasion ratio of 15%. In addition, although M7 evasion is the most challenging situation, where we independently change every part of a phishing URL to benign, BPE still shows relatively high F-1 scores (0.803-0.861). This is because each part of a benign URL connected to a phishing URL is not likely to have high similarity scores (since these are from different benign URLs), so the phishing URL has a low similarity to each newly connected vertex. Therefore, BPE with our novel similarity-based edge potential will not predict this phishing URL as benign. On the other hand, POL using a majority voting of neighbors shows low F-1 scores (0.762-0.827) in M7 evasion.

Furthermore, we found that BPE in various evasion settings outperforms most baselines in non-evasion settings. Specifically, except for M1 with an evasion ratio of 15% and M4 (resp. M7) with an evasion ratio of 10% and 15%, the minimum F-1 score of BPE in various evasion settings is 0.847 (*i.e.,* M1 with an evasion ratio of 10%), which surpasses that of the best baseline in non-evasion settings, *i.e.,* 0.840 for RandomForest. One more important fact is that evasion incurs additional costs to the attacker. To make a domain whitelisted, for instance, the attacker should pay hosting fees and maintain the domain for a considerable amount of time without any attack campaigns or should compromise other benign web servers. Some attackers do this and switch to phishing web pages at D-Day to launch a phishing attack [15]. Even after the attacker's efforts, experiments show that our method is good at detecting those evasion cases.

Evasion case study. Fig. 7 shows eight 2-hop ego networks for a phishing URL that is randomly selected for our evasion settings. The first one shows the original network connection in our dataset. The target URL (the largest red vertex) is connected to other phishy domain and words, in which case it is straightforward to classify the target URL as phishy. In the other seven networks, however, the target URL is connected to a benign domain or/and word(s). Even after these evasions, our method correctly infers that the target URL is still phishy whereas POL and RandomForest fail to detect all the evasion cases. Our method is equipped with a sophisticated edge potential assignment mechanism whereas POL does not consider them. Our theoretical analyses in Section 5 also well supports the robust nature of our method.

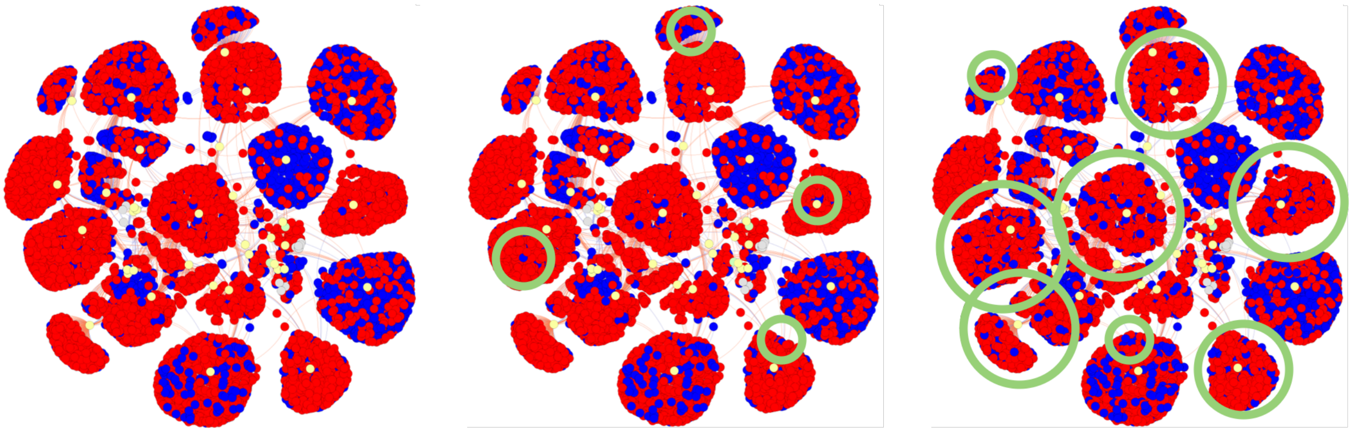

(a) Ground-truth (b) Proposed Method, *i.e.,* BPE (c) RandomForest

Figure 8: Visualization of phishy/benign predictions for a partial area in our network. RandomForest, the best inductive method in our experiments, do not use our network information so, when being projected onto it, its predictions do not strictly follow the network connectivity as shown in (c). The meaning of vertex color follows that in Fig. 3.

We also introduce other visualizations with real prediction results. Fig. 8 shows three visualizations including our method's and RandomForest's predictions. To emphasize their differences, we choose some important domain/IP/word vertices from our network and show their URL neighbors (rather than showing the full network). In Fig. 8 (a), we can observe a strong pattern that the ground-truth label follows the network connectivity in many cases. Sometimes red (phishy) and blue (benign) vertices are mixed in a cluster but this is mainly because we find the clusters in the sub-network only. Our method in Fig. 8 (b) shows a better compliance to the network connectivity than that in Fig. 8 (c). To evade our method, therefore, the majority of URLs in the same cluster should be evaded at the same time, which burdens the attacker with non-trivial costs (see our evasion cost discussion in Section 3).

7 DATA CRAWLING

To collect as many phishing URL samples as possible, we had monitored phishtank.com for a couple of months while searching other researchers' available data. There are several online datasets — many of them were released by Ma et al. who had published several papers for phishing URL detection [30, 31]. However, their data does not include raw string patterns. We also contacted them but they replied that they cannot share the raw data. Mohammad et al. also released their data in https://archive.ics.uci.edu/ml/datasets/Phishing+Websites but they also do not release their raw data used for their research [36, 37]. As mentioned earlier, we need string patterns of phishing URLs so we couldn't utilize all the above mentioned data.

Therefore, we programmed a web crawler using an automated web browser library and collected all the URLs reported for Bank of America, eBay, and PayPal. For retrieving additional information from virustotal.com, we received an academic license to their APIs and collected many such information we listed in the main paper. The academic license was activated for three months so it was more than enough for us to retrieve all the needed information.

8 CONCLUSIONS & FUTURE WORK

Although many (machine learning) methods have been proposed to detect phishing URLs, it had been overlooked that attackers can use evasion techniques to neutralize them. In this paper, we tackled the significant problem of detecting phishing URLs after evasion. After segmenting URLs into words and creating a heterogeneous network that consists of cross-related entities, we performed the belief propagation equipped with our customized edge potential mechanism which is our main contribution. Furthermore, we showed that our design is theoretically robust to evasion. We collected recent URLs and downloaded other two datasets for extensive experiments. Our experiments with about 500K URLs verify that our method is the most effective in detecting phishing URLs and also is the most robust to evasion than all baselines. Besides, we expect that our method can be easily applied to address any similar network-based problem (*e.g.,* detecting fake accounts in social networks and email spam) if it can be represented as a classification on graphs.

In the future, we will study a string and content-based robust detection method. For some evasion techniques, it is limited to only string-based detection methods. However, it requires non-trivial efforts to collect web page contents. Therefore, we think that hybrid methods will be the most useful for real-world applications.

ACKNOWLEDGMENT

The work of Sang-Wook Kim was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2022-00155586) and by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2018R1A5A7059549). The work of Noseong Park was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (No. 2020-0-01361, Artificial Intelligence Graduate School Program (Yonsei University)).

Appendix A BASELINE LEXICAL, HOST, AND DOMAIN FEATURES

We did an extensive literature survey and collected 19 features from the papers mentioned in our related work section. The complete list of the features we used in our experiments (sorted by the feature importance extracted from RandomForest, the best performing feature-based classifier in our experiments) is as follows:

(Ranking 1st) *Kullback-Leibler (KL) divergence*
The Kullback-Leibler (KL) divergence is a popular metric to measure the similarity between two probability distributions. We can calculate the KL divergence on character distributions between a URL and the English language. The reference for the character distribution in the English language is obtained from [34].

(Ranking 2nd) *Entropy of URL*
It was found that the string entropy of a URL is an important feature, since many phishing URLs have random text, causing their entropies to be higher than those of benign URLs.

(Ranking 3rd) *Digit/Letter Ratio in the whole URL*
The ratio of digits w.r.t letters in the whole URL is also important.

(Ranking 4th) *Top-level domain numbers in path*
Attackers often try to impersonate legitimate websites by adding multiple top-level domains in the path of a URL. If the count of top-level domains in the path exceeds one, then it is likely to be phishy.

(Ranking 5th) *The number of dashes in path*
This is to count the occurrence of '-' in the path. Many dashes indicate phishing URLs.

(Ranking 6th) *Blacklist*
A blacklist contains a set of malicious domains and IP addresses. If a URL has such a domain or an IP address, it can be immediately predicted as phishy. However, it is often incomplete and there are many missing malicious domains and IP addresses. In general, we do not use a whitelist because attackers sometimes compromise whitelisted servers to implant their phishing web pages and we cannot always trust whitelisted ones.

(Ranking 7th) *Length of URL*
Attackers may use long URLs to mask the phishy appearance of phishing URLs. The length of URLs plays an important role in distinguishing phishing URLs from benign ones. We use the same length standards of [36] as follows:

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

(Ranking 8th) *Presence of digits in domain*
Benign URLs do not have digits in the domain. The presence of digits in the domain is a common characteristic of phishing URLs. We set this feature as true if any digits are encountered in the domain name part of the URL.

(Ranking 9th) *Frequency of suspicious words*
We keep track of the frequencies of suspicious and most common words occurring in URLs. We choose several suspicious words like 'confirm', 'account', 'signin', 'update', 'logon', 'cmd', and 'admin'. These words are selected after surveying the literature and real-world datasets including ours.

(Ranking 10th) *Multiple sub-domains*
[36] stated the criteria for classifying a URL as phishy based on the count of its sub-domains. If a URL's resource name part has more than three dots, then it is likely to be phishy. An example of such a URL is 'http://www.outlook.3uwin.com'.

(Ranking 11th) *Brand name modifications with '-'*
We downloaded the top-1000 most visited websites from Alexa and used them as popular brand names. Phishing URLs create similar names with prefixes or suffixes. For example, 'microsoft-x.com' and 'x-microsoft.com' are phishing URLs.

(Ranking 12th) *Very long hostname*
Too long hostname typically indicates phishyness. If the length of a hostname is longer than 22, then it is phishy.

(Ranking 13th) *Prefix or suffix separated by '-' to domain*
It is well known that phishing URLs tend to add prefixes or suffixes separated by '-' to their domain to lure users into believing that the website is legitimate. For instance, an attacker may use Amazon's domain separated by a prefix as 'http://www.hello-amazon.com'.

(Ranking 14th) *Frequency of punctuation symbols*
We count the occurrence of symbols like '.', '!', '&', ',', '#', '$', and '%'; [50] observed a high percentage of punctuation symbols in phishing URLs.

(Ranking 15th) *The number of ':' in hostname*
The number of ':' in the hostname part also implies phishyness. In particular, this is used for port number manipulation.

(Ranking 16th) *Using Internet Protocol (IP) address*
Usage of IP addresses in place of domain names usually indicates fraudulent websites. For example, 'http://120.10.10.8/login.php' is most likely a phishing URL. Some attackers may use hexadecimal numbers in the domain part, *e.g.,* 'http://0x78.0xA.OxA.8'.

(Ranking 17th) *Vowel/Consonant ratio in hostname*
This feature is to calculate the ratio of total vowels to total consonants in the hostname part. Phishing URLs do not follow the standard ratio.

(Ranking 18th) *Very short hostname*
If hostname is very short (*e.g.,* smaller than five), then it is an indicator of phishyness.

(Ranking 19th) *Existence of '@' symbol*
Attackers can use '@' symbol to trick users by exploiting the property of browsers to ignore everything before '@' in the address bar. Attackers can use URLs such as 'http://www.google.com@atc.com' which causes the browser to ignore 'www.google.com' and proceed to 'atc.com'.

REFERENCES

[1] Sadia Afroz and Rachel Greenstadt. 2011. Phishzoo: Detecting phishing websites by looking at them. In *IEEE International Conference on Semantic Computing (IEEE ICSC)*. 368–375.

[2] Ankesh Anand, Kshitij Gorde, Joel Ruben Antony Moniz, Noseong Park, Tanmoy Chakraborty, and Bei-Tseng Chu. 2018. Phishing URL detection with oversampling based on text generative adversarial networks. In *IEEE International Conference on Big Data (Big Data)*. 1168–1177.

[3] Alejandro Correa Bahnsen, Eduardo Contreras Bohorquez, Sergio Villegas, Javier Vargas, and Fabio A González. 2017. Classifying phishing URLs using recurrent neural networks. In *Proceedings of the APWG Symposium on Electronic Crime Research (APWG eCrime)*. 1–8.

[4] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4.

[5] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. 2008. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment* 2008, 10 (2008), P10008.

[6] Aaron Blum, Brad Wardman, Thamar Solorio, and Gary Warner. 2010. Lexical feature based phishing URL detection using online learning. In *Proceedings of the ACM Workshop on Artificial Intelligence and Security (AISec)*. 54–60.

[7] Duen Horng Chau, Carey Nachenberg, Jeffrey Wilhelm, Adam Wright, and Christos Faloutsos. 2011. Polonium: Tera-scale graph mining and inference for malware detection. In *Proceedings of the SIAM International Conference on Data Mining (SDM)*. 131–142.

[8] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[9] Colin Cooper, Sang Hyuk Lee, Tomasz Radzik, and Yiannis Siantos. 2014. Random walks in recommender systems: exact computation and simulations. In *Proceedings of the International Conference on World Wide Web (WWW)*. 811–816.

[10] Ravindu De Silva, Mohamed Nabeel, Charith Elvitigala, Issa Khalil, Ting Yu, and Chamath Keppitiyagama. 2021. Compromised or {Attacker-Owned}: A Large Scale Classification and Study of Hosting Domains of Malicious {URLs}. In *Proceedings of the USENIX Security Symposium (USENIX Security)*. 3721–3738.

[11] Ahmad et al. 2017. https://github.com/faizann24/Using-machine-learning-to-detect-malicious-URLs.

[12] Mohammed Nazim Feroz and Susan Mengel. 2015. Phishing URL detection using URL ranking. In *IEEE International Congress on Big Data (Big Data)*. 635–638.

[13] W Nelson Francis and Henry Kucera. 1979. Brown corpus manual. *Letters to the Editor* 5, 2 (1979), 7.

[14] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. 2018. Large-scale learnable graph convolutional networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data mining (KDD)*. 1416–1424.

[15] Anti-Phishing Working Group. 2018. APWG Phishing Attack Trends Reports. https://www.antiphishing.org/resources/apwg-reports.

[16] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data mining (KDD)*. 855–864.

[17] Haibo He, Yang Bai, Edwardo A Garcia, and Shutao Li. 2008. ADASYN: Adaptive synthetic sampling approach for imbalanced learning. In *IEEE International Joint Conference on Neural Networks (IEEE WCCI)*. 1322–1328.

[18] Grant Ho, Asaf Cidon, Lior Gavish, Marco Schweighauser, Vern Paxson, Stefan Savage, Geoffrey M Voelker, and David Wagner. 2019. Detecting and characterizing lateral phishing at scale. In *Proceedings of the USENIX Security Symposium (USENIX Security)*. 1273–1290.

[19] Jiwon Hong, Taeri Kim, Jing Liu, Noseong Park, and Sang-Wook Kim. 2020. Phishing url detection with lexical features and blacklisted domains. In *Adaptive Autonomous Secure Cyber Systems*. 253–267.

[20] David J Ketchen and Christopher L Shook. 1996. The Application of Cluster Analysis in Strategic Management Research: An Analysis and Critique. *Strategic Management Journal* 17, 6 (1996), 441–458.

[21] Issa M Khalil, Bei Guan, Mohamed Nabeel, and Ting Yu. 2018. A domain is only as good as its buddies: Detecting stealthy malicious domains via graph inference. In *Proceedings of the ACM Conference on Data and Application Security and Privacy (ACM CODASPY)*. 330–341.

[22] Melissa Kilby. 2017. https://github.com/incertum/cyber-matrix-ai/tree/master/Malicious-URL-Detection-Deep-Learning.

[23] Hung Le, Quang Pham, Doyen Sahoo, and Steven CH Hoi. 2018. URLNet: Learning a URL representation with deep learning for malicious URL detection. *arXiv preprint arXiv:1802.03162* (2018).

[24] Quoc Le and Tomas Mikolov. 2014. Distributed representations of sentences and documents. In *Proceedings of the Annual International Conference on Machine Learning (ICML)*. 1188–1196.

[25] Jaehoon Lee, Jinsung Jeon, Sheo Yon Jhin, Jihyeon Hyeong, Jayoung Kim, Minju Jo, Kook Seungji, and Noseong Park. 2022. LORD: Lower-Dimensional Embedding of Log-Signature in Neural Rough Differential Equations. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[26] Yeon-Chang Lee, Nayoun Seo, Kyungsik Han, and Sang-Wook Kim. 2020. Asine: Adversarial signed network embedding. In *Proceedings of the international ACM SIGIR Conference on Research and Development in Information Retrieval (ACM SIGIR)*. 609–618.

[27] Yeon-Chang Lee, Nayoun Seo, and Sang-Wook Kim. 2020. Are negative links really beneficial to network embedding? in-depth analysis and interesting results. In *Proceedings of the ACM International Conference on Information and Knowledge Management (ACM CIKM)*. 2113–2116.

[28] Guillaume Lemaître, Fernando Nogueira, and Christos K Aridas. 2017. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *The Journal of Machine Learning Research* 18, 1 (2017), 559–563.

[29] Tian Lin, Daniel E Capecci, Donovan M Ellis, Harold A Rocha, Sandeep Dommaraju, Daniela S Oliveira, and Natalie C Ebner. 2019. Susceptibility to spear-phishing emails: Effects of internet user demographics and email content. *ACM Transactions on Computer-Human Interaction (TOCHI)* 26, 5 (2019), 1–28.

[30] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2009. Beyond blacklists: learning to detect malicious web sites from suspicious URLs. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data mining (KDD)*. 1245–1254.

[31] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. 2009. Identifying suspicious URLs: an application of large-scale online learning. In *Proceedings of the Annual International Conference on Machine Learning (ICML)*. 681–688.

[32] Pratyusa Manadhata, Sandeep Yadav, Prasad Rao, and William Horne. 2014. Detecting malicious domains via graph inference. In *Proceedings of the Workshop on Artificial Intelligent and Security Workshop (AISec)*. 59–60.

[33] Jian Mao, Wenqian Tian, Pei Li, Tao Wei, and Zhenkai Liang. 2017. Phishing-alarm: Robust and efficient phishing detection via page component similarity. *IEEE Access* 5 (2017), 17020–17030.

[34] MEC and Cornell University. 2004. https://www.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html.

[35] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781* (2013).

[36] Rami M Mohammad, Fadi Thabtah, and Lee McCluskey. 2012. An assessment of features related to phishing websites using an automated technique. In *Proceedings of the International Conference for Internet Technology and Secured Transactions (ICITST)*. 492–497.

[37] Rami M Mohammad, Fadi Thabtah, and Lee McCluskey. 2014. Predicting phishing websites based on self-structuring neural network. *Neural Computing and Applications* 25, 2 (2014), 443–458.

[38] Amirreza Niakanlahiji, Bei-Tseng Chu, and Ehab Al-Shaer. 2018. PhishMon: a machine learning framework for detecting phishing webpages. In *IEEE International Conference on Intelligence and Security Informatics (ISI)*. 220–225.

[39] Daniela Oliveira, Harold Rocha, Huizi Yang, Donovan Ellis, Sandeep Dommaraju, Melis Muradoglu, Devon Weir, Adam Soliman, Tian Lin, and Natalie Ebner. 2017. Dissecting spear phishing emails for older vs young adults: On the interplay of weapons of influence and life domains in predicting susceptibility to phishing. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI)*. 6412–6424.

[40] Chengwei Peng, Xiaochun Yun, Yongzheng Zhang, and Shuhao Li. 2018. MalShoot: shooting malicious domains through graph embedding on passive DNS data. In *International Conference on Collaborative Computing: Networking, Applications and Worksharing*. 488–503.

[41] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data mining (KDD)*. 701–710.

[42] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. 2004. Are loss functions all the same? *Neural computation* 16, 5 (2004), 1063–1076.

[43] Sam T Roweis and Lawrence K Saul. 2000. Nonlinear dimensionality reduction by locally linear embedding. *science* 290, 5500 (2000), 2323–2326.

[44] Doyen Sahoo, Chenghao Liu, and Steven CH Hoi. 2017. Malicious URL detection using machine learning: A survey. *arXiv preprint arXiv:1701.07179* (2017).

[45] Hossein Shirazi, Bruhadeshwar Bezawada, Indrakshi Ray, and Charles Anderson. 2019. Adversarial sampling attacks against phishing detection. In *IFIP Annual Conference on Data and Applications Security and Privacy (DBSec)*. 83–101.

[46] Enrico Sorio, Alberto Bartoli, and Eric Medvet. 2013. Detection of hidden fraudulent urls within trusted sites using lexical features. In *Proceedings of the International Conference on Availability, Reliability and Security (ARES)*. 242–247.

[47] James H Stock and Mark W Watson. 2015. Introduction to econometrics (3rd updated edition). *Age (X3)* 3, 0.22 (2015).

[48] Vladimir Vapnik. 1999. *The nature of statistical learning theory*. Springer science & business media.

[49] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. 2017. Graph attention networks. *arXiv preprint arXiv:1710.10903* (2017).

[50] Rakesh Verma and Keith Dyer. 2015. On the character of phishing URLs: Accurate and robust statistical learning classifiers. In *Proceedings of the ACM Conference on Data and Application Security and Privacy (ACM CODASPY)*. 111–122.

[51] Colin Whittaker, Brian Ryner, and Marria Nazif. 2010. Large-scale automatic classification of phishing pages.

[52] Hyunsik Yoo, Yeon-Chang Lee, Kijung Shin, and Sang-Wook Kim. 2022. Directed Network Embedding with Virtual Negative Edges. In *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM)*. 1291–1299.