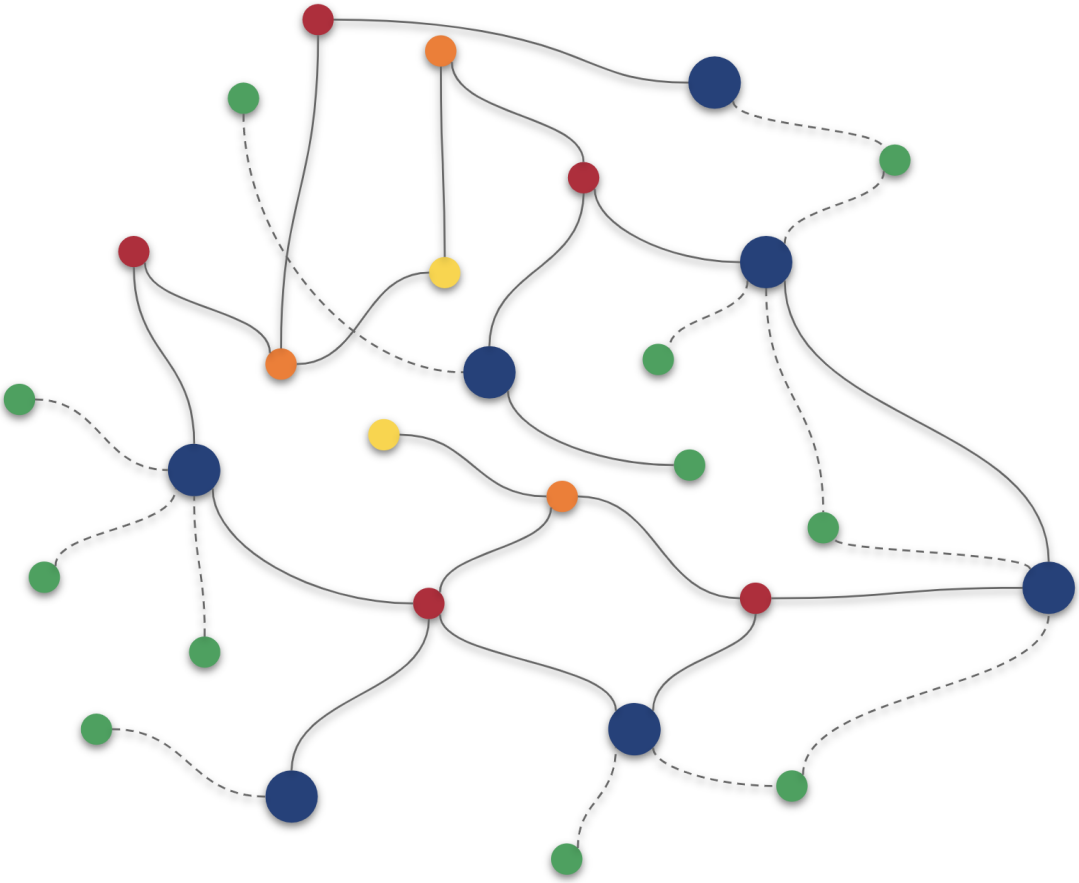


Gone Phishin'

An Investigation of Node Classification in Graphical Models for Domain Abuse Detection

Master's thesis in Complex Adaptive Systems

Joel Rosko
William Truvé

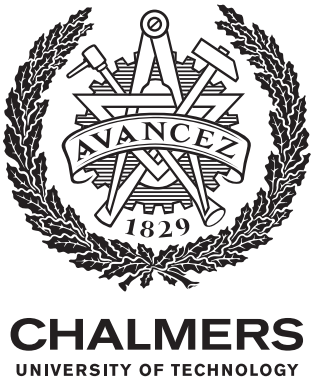
DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Gone Phishin'

An Investigation of Node Classification in Graphical Models for Domain Abuse Detection

Joel Rosko
William Truvé



Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2023

Gone Phishin'
An Investigation of Node Classification in Graphical Models for Domain Abuse Detection
Joel Rosko
William Truvé

Supervisor: Anders Hansson, Recorded Future
Examiner: Mats Granath, Department of Physics

Master's Thesis 2023
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Abstract

In today's digital era, cyber attacks pose a constant threat as attackers attempt to access proprietary data and disrupt operations on a daily basis. Phishing remains their number one attack method where users are tricked into entering sensitive information which attackers later will use or sell. The use of domain abuse detection algorithms restricts the range of attack possibilities. Furthermore, since an attack may begin as soon as a domain goes live, finding and evaluating domains quickly is of paramount importance when countering cyber threat actors.

As of now, several feature based classifiers exist and are showing good results in detecting domain abuse. However, the results are dependent on a large set of features, complicated to interpret, and struggles to generalize as attack patterns change. In this thesis we compare feature based classifiers with our implementation of belief propagation to evaluate if the use of structural information and less domain specific features can create a more interpretable and general solution.

By constructing a bidirectional graph connecting autonomous system numbers, classless inter-domain routing blocks, IP addresses, domains, and tokens extracted from the URL string, a high connectivity between nodes to propagate inference is achieved. We experiment with various techniques when initiating the graph to find an appropriate setup for belief propagation.

Our implementation of belief propagation achieves an accuracy of 91% on the entire dataset which is worse than random forest having an accuracy of 94%, however with a smaller sample of false positives. With an AUC of 0.95 the classes are well distinguishable and when optimizing thresholds and allowing nodes to be classified as "unkown", the accuracy increases to 96%.

Overall, our findings demonstrate the potential to use belief propagation for accurately identifying suspicious domains at scale, providing a valuable tool in the fight against cyber threats.

Keywords: phishing, random forest, belief propagation, loopy belief propagation

Acknowledgements

We would like to thank everyone at Recorded Future who has contributed to our master thesis by engaging in discussions regarding our work. As a special thanks, we would like to express our heartfelt gratitude to our supervisor Anders Hansson who has been a great mentor during every step of this project.

William Truvé, Gothenburg, June 2023
Joel Rosko, Gothenburg, June 2023

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AUC	Area under the ROC Curve
ASN	Autonomous system number
BPE	Byte-pair encoding
BP	Belief propagation
BI-LSTM	Bi-directional Long short-term memory
CBOW	Continous bag-of-words
CIDR	Classless Inter-Domain Routing
CSS	Cascading Style Sheets
DBSCAN	density-based spatial clustering of applications with noise
DGA	Domain Generation Algorithm
DNS	Domain name system
DOM tree	Document object model tree
GDPR	general data protection regulation
GNN	Graph neural network
HTML	HyperText Markup Language
ML	Machine learning
RFC	Random forest classifier
ROC	receiver operating characteristic
RNN	Recurrent neural network
SSL	Secure Sockets Layer
URL	Uniform Resource Locator

Contents

List of Figures

List of Tables

1
Introduction

Domain abuse is a highly relevant topic in today's society. This thesis investigates and evaluates a few different approaches on how to detect new phishing domains to prevent potential attacks.

1.1 Purpose & Aim

A recent study performed over a 6-month span in 2022 found more than 255 million phishing attempts during that period of time [1], this number corresponds to a 61% increase from the previous year. These attacks are not only increasing in number but also in sophistication, making it difficult for the victim to tell whether it's a scam or not. Machine learning (ML) models have become a common approach to solve the problem of identifying and warning a user about malicious domains by using domain name system (DNS) data as features when classifying suspicious domains.

Existing feature-based classifiers have reached promising results around 98% accuracy towards their specific datasets, however, when exposed to new data their performance will likely decrease, due to constant changes in attack patterns [2]. Aiming to establish a more robust and performant classifier, a graph schema between domains and IPs is created combined with passive data. The graph is used to propagate reputation between neighboring nodes as a way of utilizing the important structural information [3].

1.2 Problem statement

In light of the presented purpose, we aim to:

- Develop a graph-based model to classify phishing domains.
- Compare feature-only models and graph-based models.
- Minimize falsely predicted malicious domains to avoid alert overflow.

1.3 Disposition

A description of what is included in each chapter.

Introduction

Introduces the ongoing problems with phishing attacks and the weaknesses of many current solutions. We briefly describe our approach as an alternative solution and the problem statements we wish to answer to answer to evaluate the benefits of a graph-based solution.

Background

The background chapter gives a short introduction to the field of cybersecurity, mainly focusing on phishing attacks. Recorded Future as an organization is introduced and how this study could assist in their work. Lastly, the data sources and their features are described.

Related work

A lot of research has been done in the field of detecting domain abuse. In related work, multiple previous projects used as inspiration are mentioned. The mentioned projects include simple URL-string models, feature classifiers, belief propagation (BP) approaches, that highly inspired our project, and more complex neural network solutions for possible future work.

Method

Divided into two parts, the method chapter describes the selection and implementation of a feature-based classifier and the development of our BP algorithm. The chapter provides an in-depth explanation of algorithm modifications and various experimental graph constructions.

Results

The result chapter includes all results obtained when creating the random forest and BP model. The BP results contain several experiments when iterating through alternative solutions. The last sub-chapter then highlights the most optimal solution, the final model, and thoroughly presents its performance.

Discussion

The results and problems introduced previously in the report are combined to discuss the initial problem statements. The chapter also includes additional thoughts and problems encountered during the project.

Conclusion

The last chapter is a short and concise explanation of the main insights found throughout the project.

1.4 Contribution

The thesis contains three main contributions, compared to previously developed BP solutions. The first contribution is a method to handle underflow when propagating belief through a network with high-degree nodes: by averaging the messages received, numerical issues are avoided. The second contribution is to add a new kind of connection between domains: the end graph is equipped with edges between domains if the Levenshtein distance between those URLs is short. The third contribution is to extract the most common URL sub-strings using byte-pair encoding tokenization to increase graph connectivity and discover new connections.

2
Background

In this chapter, we motivate why this work is highly relevant. Furthermore, we present and explain this works relevance for Recorded Future.

2.1 Cyber security

Cybersecurity is the practice of protecting computer systems, networks, and data from unauthorized access. As most companies strive for increased digitalization and automation, cybercriminals are able to pose a great threat to all kinds of infrastructure. Moreover, cyberattackers are becoming more sophisticated in their techniques and tools, which means that businesses must stay updated on the latest potential threats and best practices to protect themselves. With the rise of large language models, cybercriminals now have the potential to train AI chatbots to constantly send out phishing emails and launch malicious websites in order to infiltrate networks [4, 5].

A model that is frequently used within the cybersecurity industry is the "kill chain" model. The kill chain is a framework that describes the different stages of how a cyber attack is prepared and launched. Understanding the process of an attack can help organizations respond to possible threats in an effective manner. Following is a description of the different steps, as well as an illustration of the kill chain in figure (2.1).

Reconnaissance: The attacker gathers information about the target.

Weaponization: The attacker creates a weapon, such as malware, to exploit a vulnerability.

Delivery: The attacker delivers the weapon to the target, typically through email or a compromised website.

Exploitation: The weapon is activated, and the attacker gains access to the target's system.

Installation: The attacker installs a backdoor or other malware on the target's system to maintain access.

Command and Control: The attacker establishes a command and control channel to communicate with the malware on the target's system.

Actions on Objective: The attacker achieves their objective, which could be stealing data or disrupting operations.

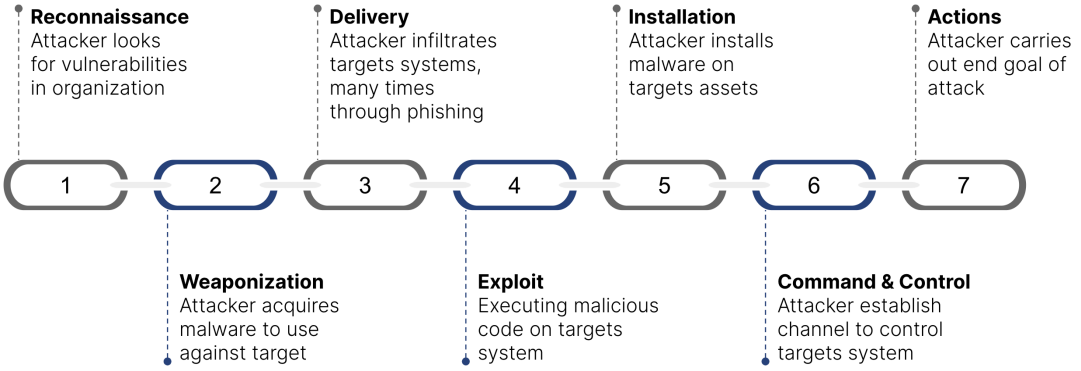


Figure 2.1: Illustration of the Cyber Kill Chain and how the different steps create a framework to carry out a large-scale attack. By hindering phishing attacks, the delivery step is obstructed.

Early in the chain, attackers will attempt to infect their targets with malware. By finding and blocking phishing websites, organizations can prevent users from falling victim to some of these attacks. This can be done through various methods, such as using email filters to block phishing emails or using web filters to block access to known phishing websites.
In conclusion, cyber security is a critical aspect of our daily lives, and understanding the kill chain and how to prevent phishing attacks can help organizations and individuals to better protect themselves from cyber-attacks.

2.2 Recorded Future

Recorded Future, founded in 2009, is the world's largest intelligence company with customers in the private sector as well as governments.[1] By continuously collecting and analyzing data published on the open web, dark web, and other technical sources, Recorded Future is able to detect existing threats and assist its customers by providing them with information relevant to their company or organization. Moreover, Recorded Future collects all newly registered domains, which presents an opportunity for early phishing detection.

With threat actors constantly working to bypass existing security measures, information about what is occurring outside the firewalls is of vital importance to adapt to current threats. The threats are detected by specific machine learning models created by Recorded Future. However, in order to extend their insight into the threat landscape they must continuously develop additional methods and models. Having the capability to quickly and with high accuracy evaluate domains being abused for phishing is one such field. Customers could then be alerted when they are being targeted, providing them with crucial information in order to avoid security breaches.

[1] https://www.recordedfuture.com/company

2.3 Data origin

The data that has been used in this project is gathered from three different sources, Tranco, Phishtank, and Recorded Future, creating a dataset containing both malicious and benign domains.

- **Tranco:** Tranco[2] is an organization that rates websites based on their traffic. A website with more traffic receives a higher ranking and is assumed to be benign.
- **PhishTank:** PhishTank[3] is a website that allows users to submit suspicious phishing domains, other users are later able to vote and validate whether or not submitted domains are used for malicious activity.
- **Recorded Future:** As described in the previous section, Recorded Future collects information on various events. Collected domains from Recorded Future are domains that were recently flagged as phishing domains.

From Tranco approximately the 100 000 top-ranked websites were collected, which serve as benign domains. The domains fetched from PhishTank and Recorded Future are merged into domains labeled as malicious. These sources yielded approximately 40 000 domains from Phishtank and 100 000 domains from Recorded Future. The combined dataset of benign and malicious domains consists of approximately 240 000 domains.

Each domain is later enriched with multiple features which are utilized by the models. The two services used to collect features for the domains are the Recorded Future service SecurityTrails[4] and Urlscan.[5] SecurityTrails provides the service to fetch domain name system and IP-related data while urlscan collects data related to the DOM tree of the website corresponding to the domain.

2.4 Domain Features: DNS and Content

Domain name system features, or DNS features for short hold information describing how a website was created, and some of its connections, such as IP adress(es). All live websites have to be registered, have an active IP address, and usually, it contains some sort of content. The list of features is extensive and can tell quite a lot about a website. Below is a brief description of the most important features.

The URL string, for example “google.com”, is an important part of a domain as it allows users to connect to a website easily. Furthermore, the URL can be analyzed to find malicious domains. One case where the URL is important is when a phishing website is created as a typosquat of a benign website, following the “google.com” example, a typosquat domain could be “goggle.com”. Machine learning models have had success in finding phishing websites, using only the URL string [6][7][8].

[2]https://tranco-list.eu/
[3]https://phishtank.org/
[4]https://securitytrails.com/
[5]https://urlscan.io/

A common term for another specific subset of features is a domain's WHOIS information. For example, every domain has to be registered by an Internet domain registrar such as "GoDaddy Inc", this feature can also give information regarding the credibility of a domain since some registrars are cheaper than others and may also provide additional anonymity making it difficult to track the owner of a domain. Other features such as who registered the domain and technical contact information can also be found sometimes, however, since the General Data Protection Regulation (GDPR) was enforced in Europe this information is easier to hide for new domains [9].

Another set of interesting features is based on the content of a website, later also referred to as the document object model tree, or DOM tree for short. These content-based features can tell whether or not the website has a login form, number of javascript files, and more.

Many domains today also use encryption when transferring data between the domain and the users. One way to encrypt data is via transport layer security, abbreviated TLS which can be identified via a padlock next to the URL in your browser. This feature establishes a secure connection between your browser and the domain's server, preventing hackers from reading or modifying transferred data [10]. However, these certificates are often found on both benign and malicious websites. This is likely due to the fact that some companies offer free certificates which allow malicious domains to disguise themselves as safe without any extra cost.

Every domain must also have at least one active IP address. This feature may be interesting to analyze since when a phishing campaign is launched a lot of phishes may resolve to the same IP address, allowing us to propagate risk in that network, however, most IP addresses usually host both malicious and benign websites which must be taken into consideration. All IPs are also linked to an autonomous system number (ASN) and are maintained by an organization owning that ASN.

A detailed view showing which features this project makes use of is shown in the method section.

3
Related Work

Trying to detect malicious websites has been a hot topic for many years now, and just as researchers try to find the best detection method, hackers are similarly using their best evasion techniques. This chapter presents some of the key findings from previous research on phishing detection.

A common approach is to extract features from the URL string, and from there train a machine-learning model to classify whether or not the website is malicious [11, 12]. Even though many previous works have reported results with high accuracy, over 90%, it has been shown that many of these models do not generalize very well. Studies have found results may drop more than 30% in performance when tested on previously unseen data [13, 14].

3.1 Data

Related research articles often make use of blacklists and whitelists to construct their dataset. When extracting legitimate domains, a common approach is to use websites that have a high ranking according to some list, however, these lists may be infiltrated, which would damage the quality of the dataset [15]. A dataset that has been made in an attempt to withstand such attacks is Tranco, which is an open-source list of benign domains [16].

Moreover, in the process of building a dataset of malicious domains, a common source to use is `Phishtank.com`, which is a crowdsourced repository that allows anyone to vote whether or not a website is used for malicious activity, this data may also have some flaws since the voting can be manipulated by hackers [17]. In our dataset, we have a mix of confirmed phishes from PhishTank as well as phishes from Recorded Future's database.

3.2 Models that act on string-based features

Predicting a domain solely based on its URL string may seem impossible but it has proven to be quite successful. This can be done in a few different ways, either extracting a couple of features from the string, for example, the number of dots and dashes, algorithms that detect if it's a randomly generated string, and the presence of special symbols [18, 19]. These models have shown a high accuracy, and even though they make it more difficult for hackers to create websites, these

detection methods can easily be avoided by handcrafted domain names. Another way of detecting malicious websites based on their string is to train a language model with transformers [8]. This too has proven quite successful but may suffer the same consequences of generalizing poorly due to poor diversity in the dataset. In our work, we recognize that the URL is obviously an important feature and can tell a lot about a domain, however, it has been shown that these models often show bias towards the dataset being used for training [20].

3.3 Models that act on infrastructure-based features

Many articles have extracted a number of features based on the domain URL string, DNS data, certificates, IP addresses, and WHOIS data, in an attempt to classify domains [21, 22, 23, 24]. This allows for a rather extensive dataset with a lot more information than just the URL string. Moreover, many phishing websites prefer registrars that are cheap and may provide additional anonymity, which is a motivation to why these features may improve a model's accuracy. However, hackers are not the only ones who prefer cheap and anonymous registration options, and even though these models can easily distinguish domains that are from Tranco versus Phishtank, it becomes more difficult to separate arbitrary benign domains, such as blogs or family websites, from malicious phishing sites. Shirazi [20] highlights several shortcomings, such as the presence of bias towards the dataset, exhibited by these models. Instead, in that work, they designed features that can visualize relationships between a domain name and some critical aspects of a phishing website.

Another shortcoming of these models is the fact that GDPR and privacy services allow perpetrators to stay anonymous, which causes a great limitation on WHOIS information of a domain [25].

3.4 Models that act on content-based features

Similarly to infrastructure-based features, one can extract features based on the content of a website, such as screenshots and the DOM tree. This may include features such as the presence of login forms and brand logos. There are also more advanced features that may be extracted from a website's source code by analyzing the underlying code structure such as HTML tags, CSS styles, and JavaScript functions [7, 26, 27]. Another feature that can often be important to analyze amongst phishing domains is the favicon [28].

Using these features may provide information regarding what type of website it is. Phishing sites tend to have as little text as possible and instead make use of images in order to avoid text-based techniques [29]. Furthermore, since the most common objective of a phishing website is to steal a person's credentials there are usually several input fields.

3.5 Belief Propagation

BP is an algorithm that has been widely used for probabilistic inference in graphical models [30, 31, 32]. One of the reasons it has grown popular is that it scales really well, even with graphs that have billions of edges [33]. A paper that we have taken a lot of inspiration from is Kim et al., [17], that showed promising results using BP. This research emphasizes how different edge potentials can increase the accuracy of BP. By embedding the nodes they calculate an edge weight based on the two nodes' cosine similarity, thereby decreasing propagated risk if the nodes are not similar despite the fact that they are connected. Other researchers have for example constructed a detailed graph with many different node types such as ASN, IP address, domains, Organization, and certificates [34]. This more detailed graph also showed great results.

For smaller graphs with a low node degree, BP works fine, even though it doesn't always converge if the graph contains loops. However, graphs that have nodes with high degrees will run into numerical issues, and therefore one must either alter the graph or the algorithm. Moreover, BP can be seen as a white-box model for smaller graphs, but when the graph grows large and complex, the results are not always intuitive to the human eye [35]. Finally, many researchers who base their model of BP usually modify it to the extent that they call it something else, [36] call their model NetConf, [34] named their model MalRank. This shows that BP is not the optimal solution for these problems, but rather a good foundation for how one can tackle the problem.

3.6 Graph Neural Networks

Graph neural networks have become a popular method for node classifications [37, 38]. Similar to BP, GNN can also be seen as message passing in a graph, however, GNNs perform their message passing by training a model, whereas BP follows a determined graph structure. Both BP and GNN have difficulty with how to best structure the graph in order to get reliable results. For GNNs, it is also easier to implement domain features in the model, as it can train edge potentials and node potentials to improve the results of the GNN [39]. GNNs have also been applied to domain classification [40]. In conclusion, GNNs can be seen as a sort of hybrid model between a feature-based classifier and BP, as it can make use of features in order to determine the required graph parameters for risk propagation and node classification.

4

Theory

Our work is based on a number of theoretical concepts which are introduced in this chapter

4.1 Random Forest Classifier

Random forest is a widely used algorithm applied in various fields of classification. The algorithm is based upon multiple decision trees to generate a prediction based on the input features of the provided data samples. A decision tree is a sequential model using a sequence of logical rules to propagate through the tree until the leaf node is reached, i.e., the last node on that branch. The leaf node then assigns the new prediction based on which class most frequently reaches that node. The number of logical operations performed before arriving at the prediction is based on the tree depth, which is the maximum number of logical operations for all branches.

For each decision node, the data is partitioned into two or more subsets of data with the objective to separate the data classes as well as possible. The more the classes get separated, the more information is gained from that split, and the subsets are seen as more pure. In the left part of Figure 4.1 a simple decision tree can be seen where the decision nodes represent the logical rules to partition the data samples [41].

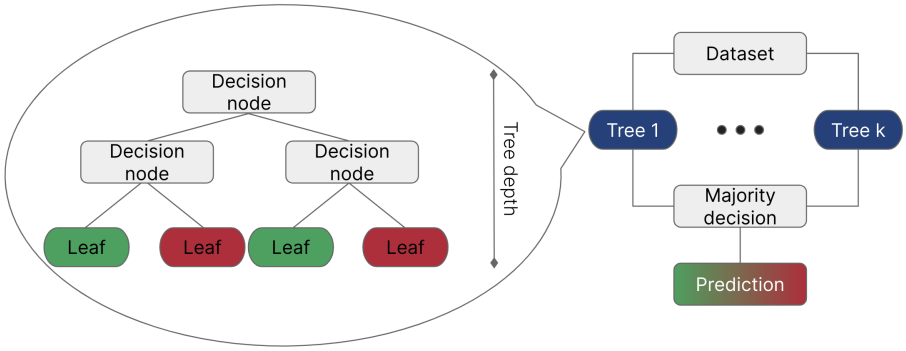


Figure 4.1: Illustration of random forest classification. The algorithm consists of multiple decision trees, where the leaf nodes correspond to that tree's prediction. The majority prediction of all trees in the forest becomes the final prediction.

As stated earlier the random forest algorithm initiates multiple independent decision trees. With each decision tree producing its own prediction, the classifier collects all

predictions and the majority class is then selected. An illustration of the random forest classifier with k decision trees can be seen in Figure 4.1 [42].

4.2 Loopy Belief Propagation

This section provides a theoretical explanation of belief propagation, using examples from our domain to give an intuitive understanding of the algorithm.
BP is a message-passing algorithm that can be applied to graph models and is used to solve inference problems. In this study, the graph contains loops which means the resulting inference will be an approximative value. Moreover, the output of the model is a belief or probability for belonging to each possible class, for each node in the graph. At the initial phase of the algorithm, each node will have a prior belief, corresponding to the initial belief of it belonging to each class prior to running the algorithm. This prior belief is often called node potential.

The message-passing part is performed such that each node sends messages to all its neighbors. One message will be sent for each class. In a case where there are three possible classes, for example criminal, police, or civilian, each node will send out three different messages to all neighbors, posing as a member of each and every class. When sending a message, the sender will also take other neighbors' messages, from the previous generation of messages, into consideration. Moreover, each node on the receiving end of the message will also act as if it belongs to every class, criminal, police, or civilian. This leads to a total of 9 messages sent, however, the receiver will only accept 3 messages, one for each label. In our implementation, the message with the minimum value will be accepted. The weight of each message is also influenced by how important the connection is between classes, policemen sending messages to criminals may for example have a weaker weight than criminals sending messages to other criminals. This weight is called edge potential.

The algorithm has three sets of variables that must be initialized, node potentials ϕ_i, edge potentials ψ_{ij}, and messages $m_{n\to i}(x_i)$. In our study, each node will have two node potentials, one for each class. The edge potentials will have four different values, given $\psi_{ij}(x_i, x_j)$ from equation (4.1) and that x_i and x_j can be either malicious or benign. In general, when $\mathbf{X}$ is the set of classes each node may be classified as, $|\mathbf{X}|^2$ gives the number of combinations for $\psi_{ij}(x_i, x_j)$.

There are a few common variations of belief propagation when calculating the messages, one can use either logarithmic values and use either min-sum or max-sum. Another variation is to use probabilistic values and use min-product or max-product [43, 44, 17]. The algorithm showcased here is with logarithmic values and min-sum.

The algorithm repeats equation (4.1) until the messages converge. If the algorithm doesn't converge, due to it alternating between two different values, it runs for a

chosen number of iterations.

$$m_{i\to j}(x_j) = \min_{x_i \in \mathbf{X}} \left[\phi_i(x_i) + \psi_{ij}(x_i, x_j) + \sum_{n \in N(i)\setminus j} m_{n\to i}(x_i) \right] \tag{4.1}$$

The expression above represents the calculation of $m_{i\to j}(x_j)$, where the minimum value is obtained by considering all possible values of x_i from the set $\mathbf{X}$. The calculation includes the contribution of various terms, such as the node potential $\phi_i(x_i)$, the edge potential $\psi_{ij}(x_i, x_j)$, and the messages $m_{n\to i}(x_i)$ from neighboring nodes n (excluding j). By selecting the value of x_i that minimizes the overall expression, $m_{i\to j}(x_j)$ is assigned the smallest possible value based on x_i.

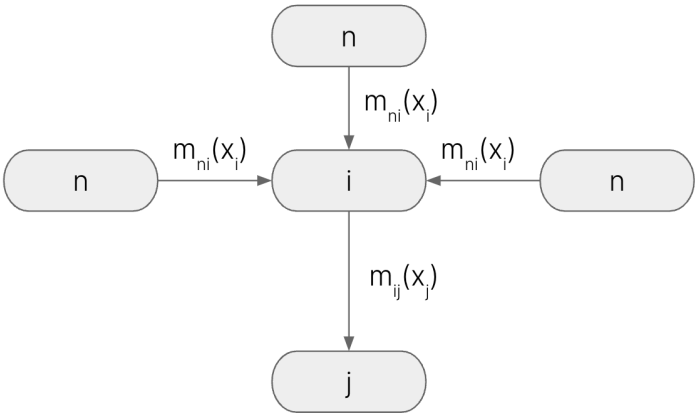


Figure 4.2: Illustration of message passing algorithm from node i to node j.

After computing $m_{i\to j}(x_j)$ for all $x_j \in \mathbf{X}$, the messages are normalized. When the message passing algorithm is complete, a final belief b is set as

$$b_i(x_i) = \phi_i(x_i) + \sum_{j \in N(i)} m_{j\to i}(x_i). \tag{4.2}$$

Similar to the normalization of messages, the final beliefs are also normalized. Moreover, the beliefs can be seen as probabilistic values which represent the probability of each node belonging to each class. In the simplest case, the end result will be binary, setting the classification of each node based on the most probable value. However, we will later show how thresholds may be added as a way to tune the end results.

4.3 Levenshtein distance

Levenshtein distance will later be used to compare string similarity between URL strings, connecting domains directly to other domains in the graph. What follows is a theoretical background of the Levenshtein distance Algorithm.

Levenshtein distance, also known as edit distance, is a measure of the difference

between two strings. Levenshtein distance is applied in a wide range of applications within the field of computational linguistics. It is defined as the minimum number of character edits needed to transform one string, string s, into the other, string t. For each edit, a cost is added to the distance, where the possible edit options are:

- **Insertion:** Cost of δ_{ins}
- **Deletions:** Cost of δ_{del}
- **Substitution:** if $[s_i \neq t_j]$, cost of δ_{sub}

The dynamic programming solution can be seen in equation (4.3) where the minimum cost, of transforming character i in string s into character j in string t, is determined. To evaluate the entire strings, the formula starts at $D(m, n)$ where $m = len(s)$ and $n = len(t)$. Next, it recursively iterates through the table of possible operations until the entire string t has been transformed. Depending on the penalties defined for the different operations the algorithm could favour some operations more than others [45].

$$D(i,j) = \begin{cases} max(i,j) & \text{if } min(i,j) = 0 \\ \min \begin{cases} D(i-1,j) + \delta_{del} \\ D(i,j-1) + \delta_{ins} \\ D(i-1,j-1) + [s_i \neq t_j] \end{cases} & \text{otherwise} \end{cases} \tag{4.3}$$

4.4 Tokenization, byte-pair encoding

When constructing the graph, the URL string will be tokenized using this particular method. What follows is a theoretical background to Byte-pair encoding.

Byte-pair encoding(BPE) is a technique used for data compression and can be applied as a sub-word tokenizer. BPE generates a vocabulary of a specified size for a given text. In other words, based on the desired vocabulary size, the algorithm creates a set of tokens (which could be letters, sub-words, or whole words) that are later mapped to the text.

To create the tokenization vocabulary, the vocabulary is initialized with all individual characters as tokens. Then, through an iterative merging of the most frequently occurring tokens, new tokens are created and added to the vocabulary. This process is repeated until the desired size of the vocabulary is accomplished [46].

> *BPE ensures that the most common words are represented in the vocabulary as a single token while the rare words are broken down into two or more subword tokens and this is in agreement with what a subword-based tokenization algorithm does.*[1]

[1] https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0

4.5 String embeddings and Word2Vec

A Word2Vec model is part of our BP, what follows here is a theoretical background.

The Word2Vec algorithm is an unsupervised learning model published by Google in 2013 [47]. Word2Vec has demonstrated several use cases for natural language processing, with applications linked to information retrieval and recommendation systems. The algorithm provides a tool to generate vector embeddings of words while still capturing semantic and syntactic relationships between words.

Word2Vec can be applied with two different modeling architectures, skip-gram, and continuous bag-of-words (CBOW). As of now, the most common approach is CBOW. With a neural network, CBOW uses the context, as well as the surrounding words, to predict the word in the middle. The context words serve as input, their current vector representations are then averaged in a projection layer. The order of the input words is not taken into consideration. The projection layer relies on a log-linear classifier to predict the most probable middle word. By specifying a window size, the number of words to include when predicting the middle word is adjusted. An example of CBOW with four words as context input is shown in Figure 4.3.

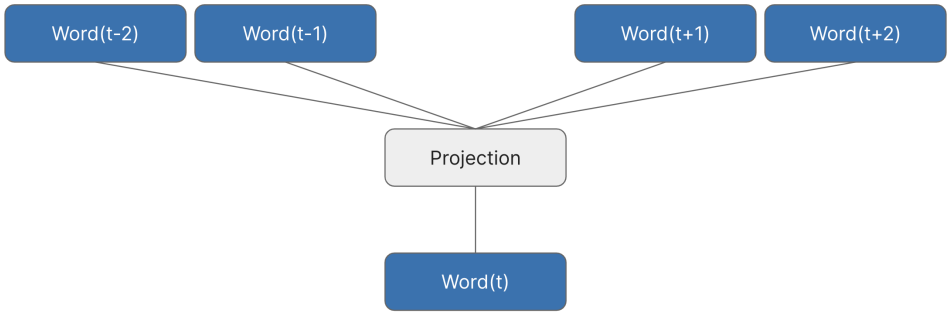


Figure 4.3: Illustration of CBOW where four words are used as context, two before and two after. The word vectors are projected into the same position, and thus their vectors are averaged. The projection layer then outputs the most probable middle word.

Once the model is trained on a text corpus, each string occurring in the corpus receives an embedding. The similarity or relationship between words could then be evaluated based on the multidimensional vector representation, by calculating cosine similarity for example. The model has proven to define clear similarities between ”big“ and ”bigger“, and more impressively relationships between ”France“ and ”Paris“ [48].

4.6 Graph embeddings and Node2Vec

Node2Vec is an algorithm that utilizes graph structure in order to create vector representations of each node. The generated node embeddings aim to be of a lower

dimension than the initial graph. The Node2Vec embeddings offer users the ability to use network data as input to traditional classifiers or evaluate similarities between nodes.

How Node2Vec manages to keep network structure of neighbouring nodes is by running several biased random walks from every node of a specified length l. Each step in the walk is determined by the probabilistic outcome, P, of equation (4.4). Here, c denotes the nodes traversed in the walk, v is the previous node, x is a possible next node, $\pi_{v,x}$ is the unnormalized transition probability between v and x, and Z is a normalizing constant. If no edge between the two nodes exists, $P = 0$.

$$P(c_i = x | c_{i-1} = v) = \begin{cases} \frac{\pi_{v,x}}{Z} & if\ (x,v) \in E \\ 0 & otherwise \end{cases} \tag{4.4}$$

A search bias α is introduced to allow the random walks to explore the absolute close proximity of the starting node, as well as traverse further away. The transition probability $\pi_{v,x}$ is therefore updated using equation (4.5). Here, t is the node traversed prior to node v, and w is the edge weight.

$$\pi_{v,x} = \alpha_{p,q}(t,x) \cdot w_{v,x} \tag{4.5}$$

The bias is defined by equation (4.6) where p and q control how quickly the walk explores the graph. While $d_{t,x}$ denotes the shortest path between node t and x. As $d_{t,x}$ represents the minimum number of edges needed to traverse from node t and ending at node x, $d_{t,x}$ could only receive a value within {0,1,2}. This is due to if $t = x$, the number edges needed are zero, if node v has to be included in the path the number of edges traversed are two, and if there exist a directly connected edges, $d_{t,x} = 1$.

$$\alpha_{p,q}(t,x) = \begin{cases} \frac{1}{p} & if\ d_{t,x} = 0 \\ 1 & if\ d_{t,x} = 1 \\ \frac{1}{q} & if\ d_{t,x} = 2 \end{cases} \tag{4.6}$$

With the generated walks, the embeddings are created using Word2Vec, which is explained in section 4.5, however this time by using the skip-gram architecture. The walk is then seen as a sentence where each node represents a word [49].

4.7 Long short-term memory (LSTM) and bi-directional LSTM

A bi-directional LSTM model will be used to assist in the classification of nodes in our graph, here, a theoretical background is presented.

Long short-term memory (LSTM) networks are a type of recurrent neural network (RNN) architecture. RNN is a powerful tool when working with sequential data, due to its use of feedback loops which pass information from the hidden states, previous

data in the sequence, to add information as the sequence proceeds. The classical RNN, however, experiences problems with vanishing gradients as the sequences get longer. LSTM network provides a solution to avoid vanishing gradients and therefore offer a better long-term memory [50].

The way LSTM models retain long sequences of information is with the help of a memory cell, shown in Figure 4.4 as cell states. What information is stored or discarded is handled by the three gates in an LSTM cell. The input gate ingests new information from the sequence and previous hidden states, using a *sigmoid* activation function to store valuable information in the cell state. The forget gate determines what information should be removed from the cell state similar to the input gate, by passing the previous hidden state into a *sigmoid* function. While the last gate, the output gate, outputs the new hidden state of the cell. The output from the input gate is passed through a *sigmoid* function and then multiplied with the *tanh* value for the cell state. The output could either be used as a prediction or serve as hidden state input to a new cell [51].

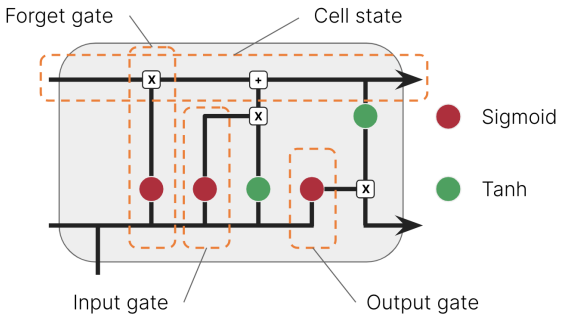


Figure 4.4: LSTM cell displaying the three important gates used to process information as well as the memory cell which stores information from the sequence.

In contrast to a regular LSTM model which iterates through the data from start to end, a bidirectional LSTM model processes the data from two directions. By incorporating two separate layers, one which processes the sequence in ordinary order while the other layer process the data from end to beginning. This allows the model to benefit from both prior and future information which has shown two be an improvement in some classification tasks. The bidirectional output is then merged to be used as input to the following cells or serve as prediction [52].

5
Methods

In this chapter, the methods used to create and fine-tune our models are described in detail. Since multiple separate components belong to the two different models, the implementations will each have their own sub-section.

The flowchart in Figure 5.1 shows on a large scale the main steps throughout the project to reach the final results. The two first points, data, and analysis are closely related, while the model developments and evaluation are more independent.

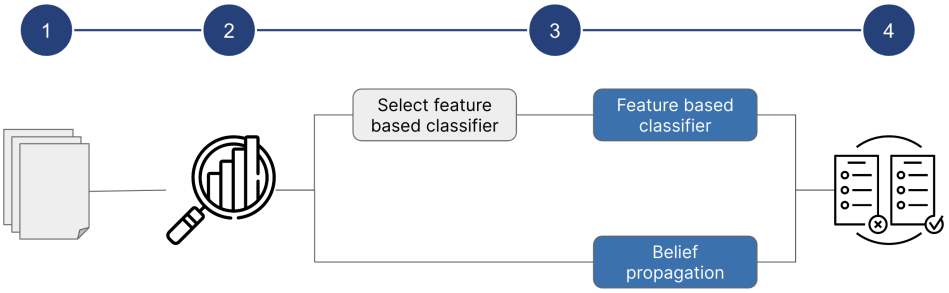


Figure 5.1: Flowchart of the steps needed to arrive at end goal. The first and second step is highly related and revolved around gathering and analyzing the data. The third step is divided into two areas, selecting and developing a feature-based classifier, and implementing the BP algorithm. The fourth and final step is then to analyze the various models' pros and cons.

5.1 Data Preparation

The data that was briefly described earlier in section 2.3 includes multiple features, each linked to one domain. As said earlier approximately 240 000 domains have been fetched. To only evaluate the more relevant domains, expired domains, i.e. domains no longer resolving to any IP address, are removed. The final dataset therefore contains around 150 000 domains, where 90 000 are labeled benign while the remaining 60 000 are labeled malicious.

A list with the enriched features and a short description can be seen below from which the feature-based classifier uses the majority of features while the BP solution includes a far smaller sample of features.

1. **Domain:** Domain URL string.

2. **IP:** List of IP or IPs which the domain resolves to.
3. **TLD:** The domains top-level domain.
4. **WHOIS registrar:** Where the domain is hosted.
5. **SSL self-signed:** Whether or not the SSL certificate is self-signed.
6. **SSL issuer organization:** Name of organization which issued SSL certificate.
7. **Number of IPs:** Number of IPs domain resolves to.
8. **DGA probability:** Probability that a domain's URL string has been generated by an algorithm.
9. **Number of dashes:** Number of dashes in URL.
10. **Number of dots:** Number of dots in URL.
11. **URL length:** Number of characters in URL.
12. **URL entropy** Entropy measure for randomness in URL.
13. **ASN:** AS number of IP.
14. **ASN owner:** Organization owning AS number.
15. **IP degree:** Number of domains resolving to the same IP.
16. **Number of CSS files:** Number of CSS files included in the website.
17. **Number of JavaScript files:** Number of JavaScript files included in the website.
18. **Number of images:** Number of image files included in the website.
19. **Has login form:** If there exists a login form on the website.
20. **Number of input forms:** Number of input forms found on the website.
21. **Label:** Class domain belongs to.

5.1.1 Label encoding

Based on the list above, several of the features are of categorical type while the majority of classifiers require numerical input. The features that had to be preprocessed, due to having too many different options, were number 3, 4, 6, and 14. These features were encoded by using a basic label encoder.[1] Separate label encoders were built for each of these features, where each possible feature label is assigned an integer representation.

5.2 Feature based classifier

There are numerous traditional classifiers available off the shelf. Therefore a simulation evaluating several of those was carried out to pick the optimal alternative, to use as a comparison model. In Table 5.1 the top five classifiers are shown, although the results are similar, the random forest classifier is highlighted due to best performance and will be used moving forward.

[1] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html

Model	Accuracy	AUC	Recall	Precision	F1
Light gradient boosting machine	0.93	0.97	0.86	0.97	0.91
K nearest neighbours	0.92	0.95	0.85	0.93	0.89
Random forest	**0.94**	**0.97**	**0.87**	**0.97**	**0.92**
SVM	0.91	0.90	0.82	0.95	0.82
Logistic regression	0.91	0.95	0.83	0.93	0.88

Table 5.1: Results for some of the evaluated classifiers to select the optimal one for the current dataset. As can be seen, the majority performs fairly similarly, however random forest has a slight edge in most evaluation metrics and will therefore be used moving forward.

5.2.1 Random forest classifier

When developing the RFC model, a Python module called scikit-learn which contains an efficient and customizable random forest package[2] was used. The model was then trained using the features mentioned in section 5.1 at the beginning. Since the classifier only accepts numerical input, all categorical features were embedded as explained in section 5.1.1. For all samples with missing values, "NaN", the values were set to -1. Without any fine-tuning, the current setup serves as our initial RFC model. Figure 5.2 briefly explains the creation of the initial model and then the steps when we try to improve its performance.

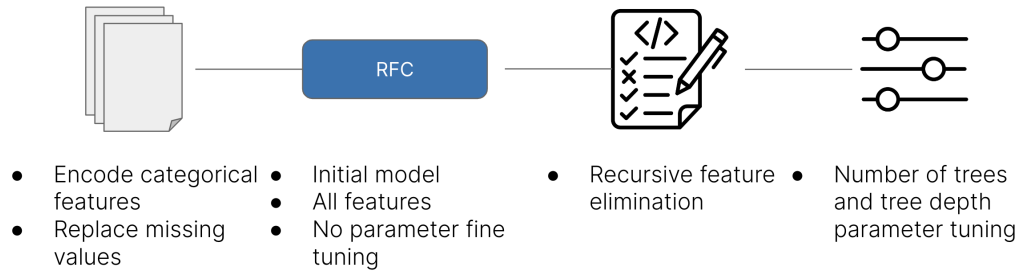


Figure 5.2: Description of the main steps when creating the initial RFC model, and the methods used to improve its performance.

With an initial model trained using all features, a feature importance simulation was carried out to find the most dominant features. Reducing the number of features simplifies the decision path and also reduces misleading information. A recursive feature elimination algorithm was applied. The algorithm removes one feature each run, the currently least important, until no features are left. Once completed only the subset of features which maximizes performance are kept [53].

The last step when fine-tuning the RFC model was to decide the optimal model parameter, number of trees, and maximum depth of trees, which is shown in Figure 5.2. Four different tree depths were used as can be seen in the list below, for each of those we changed the number of trees used, ranging from 10 to 300 trees. At each step, five runs with new data splits were executed to calculate the average.

[2]https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

- **Tree depths:** 6, 12, 24, 48
- **Number of trees**: 10–300 with steps of five

The fine-tuned RFC model will be selected based on the most performant parameter setup. Once the accuracy of the several models converges an optimal RFC model is found.

5.3 Belief Propagation

This section describes the different steps taken to run our modified BP. Moreover, a brief part of our data analysis is presented here as a motivation for the use of Levenshtein distance as well as our chosen method of tokenization.

5.3.1 Graph construction

The first step before we can apply BP is to construct a graph. We experimented with a few different node types and connections between the nodes and ended up with the following node types: Domains, Tokens, IPs, CIDR-Blocks, and ASNs. The results of a few different graph compositions is presented later in the results. Possible connections can be seen in Table 5.2. Domains will be linked to other domains if their normalized Levenshtein distance is less than 0.1. If the URL string of a domain contains a token created by the BPE-tokenizer, described in the theory section, an edge will connect those two nodes. Furthermore, each domain is connected to one or more IP addresses, and IPs resolve to a CIDR block which in turn connects to an ASN. To clarify further, every node is unique, meaning that if two domains have the same IP address or token, that token or IP will be connected to both domains and allow risk to spread via itself. An illustration of an example graph is shown in Figure 5.3. The motivation for including IP addresses as well as CIDR-Blocks and ASNs is to increase the connectivity of the graph. It has also been used by previous works on the topic [17]. The motivation for the use of Levenshtein distance and tokens is described in the following section.

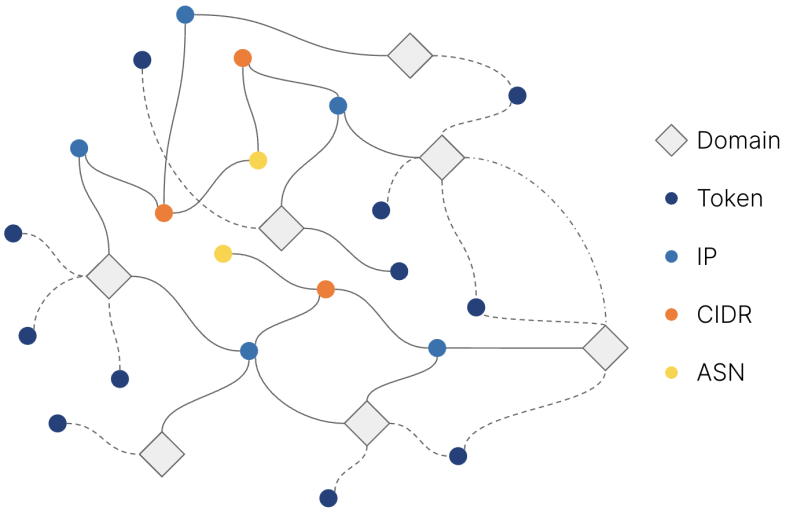


Figure 5.3: Illustration of the constructed graph with all possible node and edge types.

Table 5.2: Different node types and possible connections for each node type

Node type	Neighbours
Domain	Token, IP, Domain
Token	Domain
IP	Domain, CIDR-Block
CIDR-Block	ASN
ASN	CIDR-Block

5.3.2 Levenshtein neighbours

After performing extensive data analysis, it was found that many phishing domains have very similar names, often only differing by one or two characters. Attackers usually register multiple domains with similar names to start their attack, then if one domain is detected as malicious and shut down there are other domains still active.

Due to this, the Levenshtein distance between all domains was calculated and then normalized using equation (5.1). Here, s_1 represents one URL and s_2 the other, and l is the Levenshtein distance between the two strings. The normalized distance ranges between zero and one, where a lower value indicates similar strings.

$$1 - \frac{|s_1| + |s_2| - l}{|s_1| + |s_2|} \tag{5.1}$$

The normalized distances are used to create clusters of domains where each cluster corresponds to a set of domains referred to as Levenshtein neighbors. For two domains to be seen as neighbors their normalized Levenshtein distance must be less or

equal to 0.1. Depending on the different lengths of the URLs, this corresponds to a difference of 10% and is often equivalent to one or two string edits. For domains to be in the same cluster, however, a domain needs to be neighboring to at least one other domain in the cluster. Our clustering approach is inspired by the DBSCAN algorithm which identifies clusters based on their density [54].

Why this information could be beneficial when creating the graph and why it currently is taken advantage of can be seen in Figure 5.4. Here, all domains registered during a random sample of days were evaluated using the Levenshtein neighboring method. The domains shown are the ones corresponding to a cluster where at least one of the domains was classified by Recorded Future as a phishing website after being active for almost one month. The red bars are phishing domains while the blue bars represent clustered domains not yet convicted of phishing. For more clarification, one of the red domains is *bankofamerica-secure02[.]com* while the suspicious-looking URL *bankofamerica-secure01[.]com* is one of the blue domains. The classification of these websites does not correspond to any ground truth, it does however give an indication that Levenshtein distance may be used to find new phishing domains.

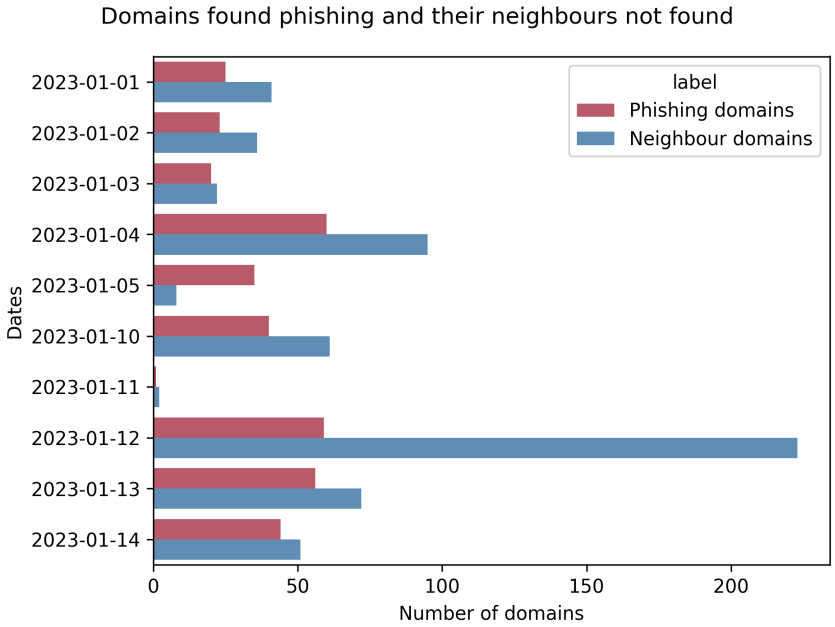


Figure 5.4: Histogram of domains registered the specific days with at least one neighboring domain and at least one domain are a phishing domain. The red bar represents domains found to be phishing domains while the blue bars are domains not said to be phishing domains.

When constructing the graph, Levenshtein edges are only added between two neighboring domains. The reason for not drawing edges between each and every domain in the cluster is that the risk will propagate throughout the chain of domains when running BP. The addition of these edges allows risk to propagate directly between two different domains, eliminating the need for intermediate nodes.

5.3.3 BPE tokenization

The BPE tokenizer was trained on the entire dataset of domains and a vocabulary size of 8000. By allowing for a rather large vocabulary, rare and specific words or sub-words are found as tokens in the URLs. However, the limit still manages to increase the connectivity of the graph since not an infinite amount of tokens are allowed.

For each URL, the tokenizer is used to extract the found tokens. Before the tokens are added to the graph, some modifications are done to the token vocabulary. We start by removing all tokens with less than four characters, $|token| < 4$. Next, we use the elbow method, the largest perpendicular distance, to find the stabilizing point which is used as a cutoff point, shown in Figure 5.5. By adding a straight line from the most frequent token to the least frequent token and calculating the perpendicular distance to the token frequency line, the largest perpendicular distance is found [17]. The words prior to the stabilizing point are deemed as stop-words and are removed, while the remaining words are added to the final vocabulary.

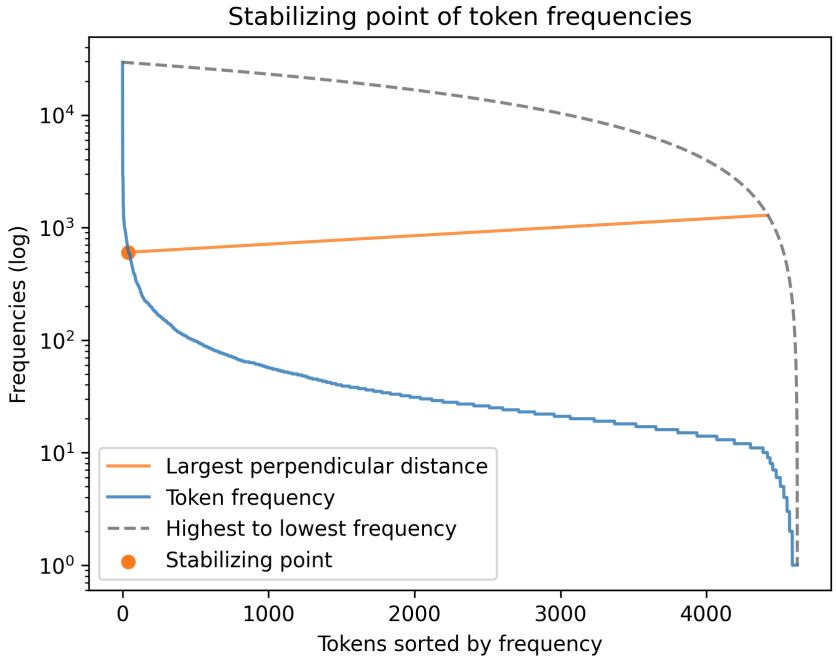


Figure 5.5: Stabilizing point, orange circle, found to use a cutoff point for stop-word removal. The point is located at the largest perpendicular distance between token frequencies and a straight line connecting the highest frequency with the lowest. Y-axis in log-scale.

All of the tokens in the final vocabulary are then added as nodes when constructing the graph. The domains in the graph which have any of those tokens in their URL receives an edge to the node associated with the token. Due to the removal of multiple tokens, some domains receive no token connection, and the final vocabulary size is approximately 4600. By adding tokens, increased connectivity between phishing

domains was achieved since many strings are reused by attackers to appear legit. This is exemplified in Figure 5.6.

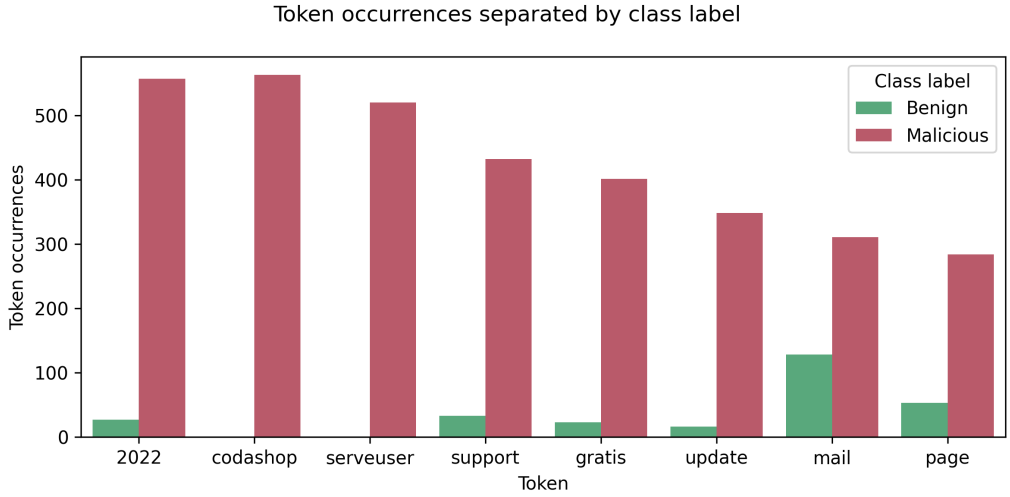


Figure 5.6: Bar plot displaying the token frequencies from the dataset for some randomly sampled tokens. The green bars represent the number of times that token was used in a benign URL while the red represents when the token was found in malicious URLs.

5.3.4 Edge potentials

To initialize the edge potentials three different approaches were used as experiments. As a first approach, we applied the more classical approach of using a coupling value, *epsilon*. The edge potential when passing messages between nodes for the same class label receives a higher value than when the class labels differ. We used an *epsilon* value of 0.3, the values can be seen in Table 5.3.

	Benign	Malicious
Benign	$0.5 + \epsilon$	$0.5 - \epsilon$
Malicious	$0.5 - \epsilon$	$0.5 + \epsilon$

Table 5.3: Compatibility matrix when using epsilon edge initialization. When passing messages between nodes for the same class label a higher potential is set than when two different class labels interfere.

	Benign	Malicious
Benign	$min(t_+,\ S_{C,n})$	$max(t_-,\ S_{C,n})$
Malicious	$max(t_-,\ S_{C,n})$	$min(t_+,\ S_{C,n})$

Table 5.4: Compatibility matrix used for similarities between node embeddings from Word2Vec or Node2Vec. The potential between two equal class labels is the minimum value of the scaled cosine similarity, $S_{C,n}$, and the maximum threshold, t_+. For opposite class labels, the minimum threshold, t_- is used.

When setting the edge potentials for the second experiment, using Word2Vec embeddings, multiple steps were carried out to reach the final potentials. The nodes represented by strings, domains, and tokens, were embedded using a trained Word2Vec[3]

[3]https://radimrehurek.com/gensim/models/word2vec.html

model to receive a vector representation of length 100. However, with the remaining possible nodes, IP, CIDR-block, and ASN, the embeddings were calculated as the average of neighboring nodes' embeddings. Equation (5.2) explains the averaged vector representation of the ith node, where the sum of neighboring node embeddings is divided by the number of neighbors. $|N(i)|$ denotes the number of neighbours node i has, whereas $N(i)_{emb}$ is the vector embeddings of the neighbours of node i.

$$\mathbf{n(i)}_{emb} = \frac{1}{|N(i)|} \sum_{\mathbf{n}_{emb} \in N(i)_{emb}} \mathbf{n}_{emb} \tag{5.2}$$

The last approach used as edge potential initialization was to use Node2Vec. A Node2Vec model[4] was trained on the entire graph, generating vector embeddings for all types of nodes in one step.

With all nodes embedded using either Word2Vec or Node2Vec, the cosine similarity between nodes, or the node embeddings, are calculated using equation (5.3). The similarities are scaled to range between zero and one by equation (5.4). The potentials are then as shown in Table 5.4, with some modifications compared to the *epsilon* potentials.

$$S_C(n(i), n(j)) = cos(\theta) = \frac{n(i)_{emb} \cdot n(j)_{emb}}{||n(i)_{emb}||\ ||n(j)_{emb}||} \tag{5.3}$$

$$S_{C,n} = \frac{S_C + 1}{2} \tag{5.4}$$

The compatibility matrix introduces two thresholds, t_+ and t_-, which control the minimum or maximum edge potential. When messages are passed between to equal class labels, the minimum value of t_+ and $S_{C,n}$ are used as potential. When two opposite class labels interfere the maximum value of t_- and $S_{C,n}$ are used.

5.3.5 Node potentials

Similar to edge potentials, two different node potential initializations were tried as experiments to improve the final model performance. The following only applies to domain nodes since all other types of nodes are initialized as unknown or unlabeled for both experiments. The first approach was to set the potentials as seen in the list below:

- **Benign:** $Benign = 0.99$, $Malicious = 0.01$
- **Unknown/Unlabeled:** $Benign = 0.5$, $Malicious = 0.5$
- **Malicious:** $Benign = 0.01$, $Malicious = 0.99$

Each node has two potentials, one for benign belief and one for malicious. For domains labeled as benign, the initial potential is clearly set as benign. The opposite applies to malicious domains. For domains without any label, the ones who are to be predicted, the potential is set to 0.5 to not favor any of the possible predictions.

[4]https://github.com/eliorc/node2vec

Another less general approach was also used when initializing the unlabeled domain nodes. Based on the predictions of the LSTM model (see below), the initial node potential was shifted according to the prediction of the LSTM model. The shift was performed using equation (5.5). The scaling reduces the value range to be between 0.25 and 0.75 to avoid the potential of favouring any of the class labels too much.

$$p_s(i) = 0.5 + \frac{p(i) - 0.5}{2} \tag{5.5}$$

An illustration of the LSTM model used to generate the predictions can be seen in Figure 5.7. The model takes a URL as an input and outputs a probability value due to the final layer being a fully connected layer with *sigmoid* as the activation function. Before passing URLs into the model, they were padded to have equal lengths. At the second layer, the strings were embedded into vector representations. Once embedded the input reaches the three connected bidirectional LSTM layers which is the main core of the model. The last LSTM layer returns a one-dimensional vector which the output layer uses to form a prediction. The architecture of the network is heavily influenced by a project posted by "jackaduma" on GitHub[5], where it reached an accuracy of 99% predicting phishing URLs.

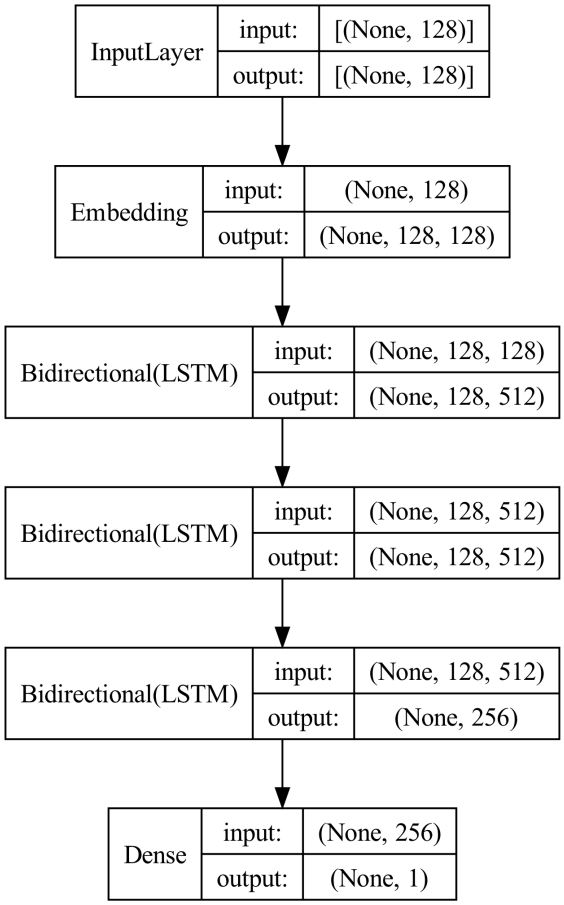


Figure 5.7: Sequential bidirectional LSTM model displayed with input sizes for each layer.

Using the model to retrieve the scaled prediction value, $p_s(i)$, is used in equation 5.6 to set both the benign and malicious potential. The larger $p_s(i)$ is, the larger the predicted maliciousness of the domain.

$$\phi_i = \begin{cases} Benign = 1 - p_s(i) \\ Malicious = p_s(i) \end{cases} \tag{5.6}$$

[5]https://github.com/jackaduma/NLP4CyberSecurity

5.3.6 Handling underflow for high degree nodes

The standard equation for BP can easily cause numerical issues for nodes with a high degree. Many researchers often handle this by removing nodes that have too large of a degree, for example by excluding tokens that are too common. We went with a slightly different approach, we still removed the most common tokens in order to avoid cluttering the graph, however, we still have nodes with high-degree, common tokens as well as some IP addresses. This caused numerical issues despite the fact that we are using logarithmic values, simply because the messages grew too negative for standard message passing BP. We, therefore, decided to average the incoming messages when performing the message passing, which is described by equation (5.7). When inferring the final belief, however, we used the standard formula, equation (5.8). This worked well numerically since the final belief was only inferred on domains, which typically do not have a very high degree.

$$m_{i\to j}(x_j) = \min_{x_i \in \mathbf{X}} [\phi_i(x_i) + \psi_{ij}(x_i, x_j) + \frac{1}{|N(i) \setminus j|} \sum_{n \in N(i)\setminus j} m_{n\to i}(x_i)] \tag{5.7}$$

$$b_i(x_i) = \phi_i(x_i) + \sum_{j \in N(i)} m_{j\to i}(x_i) \tag{5.8}$$

5.3.7 Algorithm implementation

When implementing the algorithm, we made a few more design choices. At first, the data is split, where one set of the domains kept their label of benign/malicious. The remaining set of domains had their labels dropped with the goal of predicting the label. The initial potential of the unlabeled domains was set with help from the BI-LSTM model whereas the labeled domains had a fixed starting potential. Once the graph and all the node and edge potentials had been set, the messages were initialized to the value 0 since logarithmic values were used ($0 = log(1)$). Moreover, messages are only being passed to nodes that are unlabeled, meaning that domains that already have a label do not have the ability to change their belief.

Secondly, all messages were normalized for each iteration and label. At this stage, the message passing part of the algorithm was repeated until convergency conditions had been met. Finally, a belief was inferred to the previously unlabeled domains, predicting their probability of being either malicious or benign.

6
Results

The result chapter is divided into two main sections, one presenting the results from developing the RFC model and the second presenting the results from several BP models.

6.1 RFC Results

As explained in section 5.2.1, when developing and fine-tuning the RFC model, two main steps were conducted, feature elimination and parameter selection. Figure 6.1 displays the feature importance metrics for the initial model. The importance is on the x-axis and measured as a mean decrease in impurity, the better a feature manages to separate the class labels, the purer data splits. As can be seen the number of dots in the URL clearly is the most important feature, while most other features are fairly similar in importance. The IP risk score feature consists of only missing values and therefore has no importance.

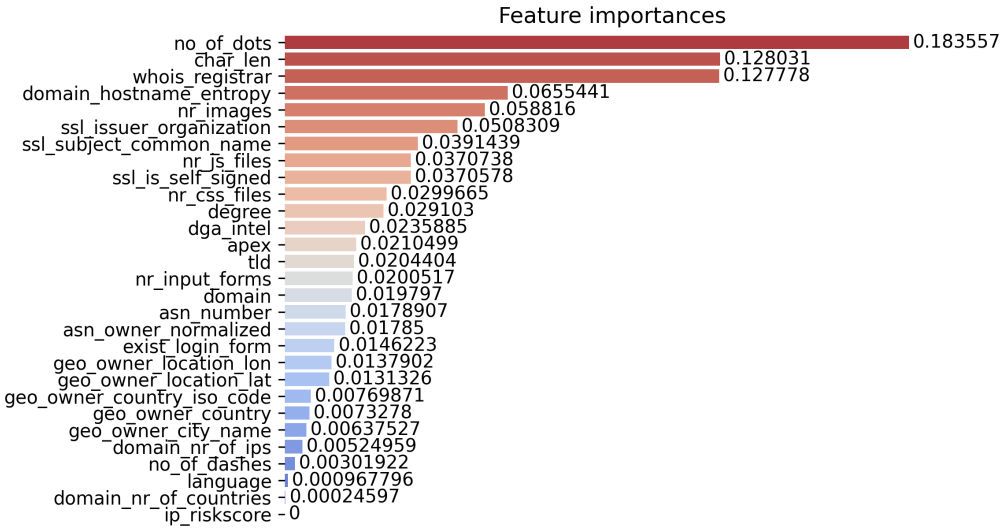


Figure 6.1: Bar plot showing the importance of the used features, with a mean decrease in impurity on the x-axis. Decreasing the impurity is a result when the feature well splits the class labels.

By running the recursive feature elimination algorithm, three features were removed, IP risk score, number of countries, and language. By studying Figure 6.1, it shows that these were the features with the least importance. Worth noting is that mean

decrease in impurity isn't an optimal measurement when exposed to high cardinality features, features containing multiple unique values. The number three ranked feature, who-is registrar, contains multiple unique values, however still ranks well compared to the other features.

The next step was to select the optimal parameter configuration as discussed in the method chapter, section 5.2.1. The result of the simulation can be seen in Figure 6.2, where the lines represent different tree depths and how the accuracy changes as more trees are added to the models. The figure shows that using either 24 or 48 as tree depth has no real impact on the accuracy of the models. The performance for those two models also seems to converge at around 75 to 100 trees, even though the accuracy overall has very small changes. To continue we selected the parameters displayed in the list below.

- **Tree depth:** 24
- **Number of trees:** 100

Using 24 as tree depth instead of 48 helps reduce model complexity, without impacting the model performance. The selection of the number of trees had less impact on final performance, however moving forward, 100 trees are used.

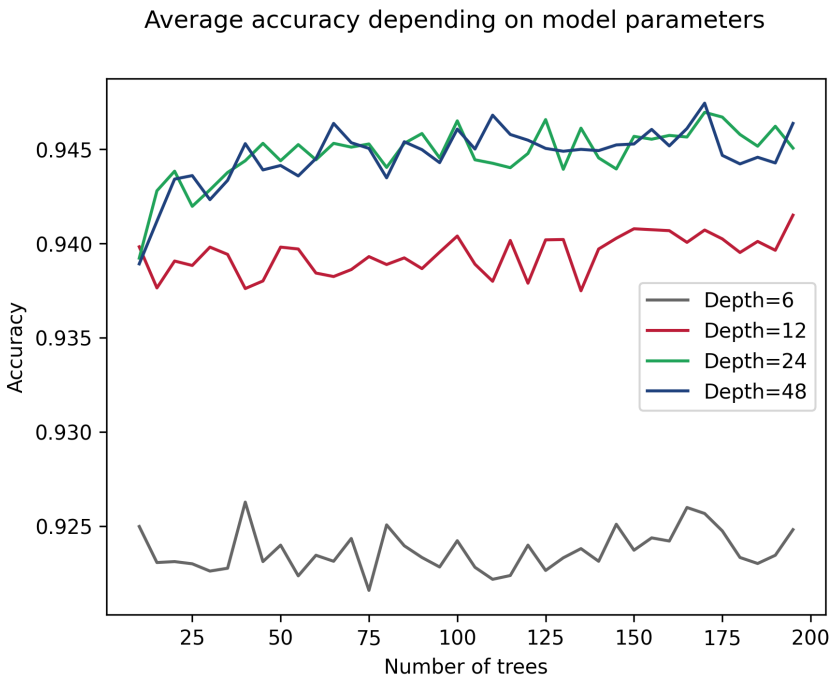


Figure 6.2: Simulation of random forest classifiers with varying tree depth and number of trees. For each step, five models were evaluated with newly generated data splits and the results shown are the average across those five simulations.

The performance of the final model is slightly improved from the simulation and reaches an accuracy of 95%, as can be seen in Table 6.1. The support column presents the number of samples from each class label in the test set, due to the dataset containing more benign domains than malicious, the test set is skewed a

bit towards benign domains. The model's precision for finding malicious domains is high, at 98%, which tells that it rarely predicts a benign domain as malicious.

	Precision	Recall	F1-Score	Support
Benign	0.93	0.99	0.96	18332
Malicious	0.98	0.88	0.92	11438

Table 6.1: Result from the final RFC model, achieving an accuracy of 95%. The other metrics are shown for both class labels, where support represents the number of samples for each class in the test set.

The model performance can also be seen in Figure 6.3, the figure shows the receiver operating characteristic (ROC) curve and the area underneath (AUC). The ROC curve, dark blue line, demonstrates the trade-off between the accurately predicted malicious domains, versus the falsely predicted malicious domains. By studying the ROC curve, how well the model manages to discriminate between class labels when exposed to a large range of classification thresholds.

The AUC score quantifies the overall performance of the model, the score represents how well the model manages to distinguish between class labels, without taking any specific classification threshold into account. The AUC score of 0.98 suggests that the classifier has great discriminative capabilities, with a strong ability to differentiate between benign and malicious instances.

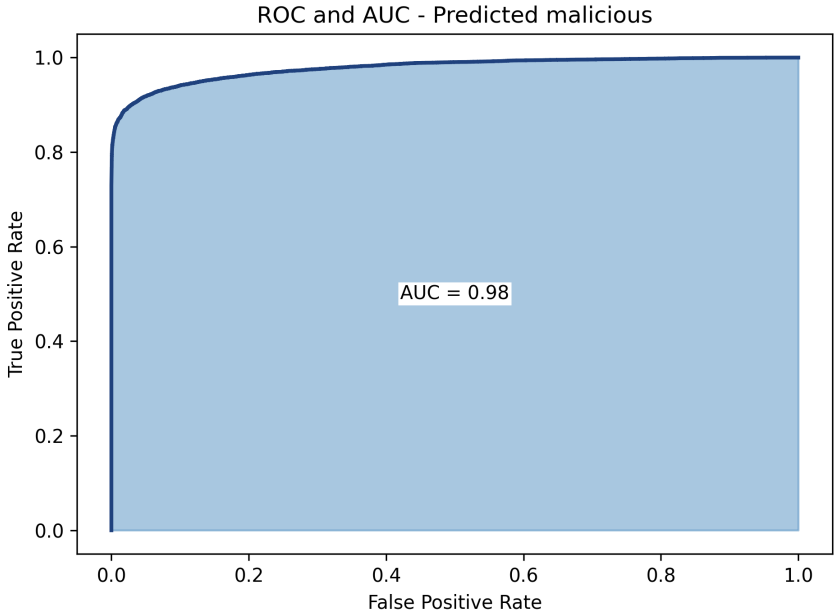


Figure 6.3: ROC curve for the final model with a true positive rate on the y-axis and false positive rate on the x-axis. The AUC score is 0.98 which indicates well-separated classes.

As a final experiment, some features were removed in order to make the features

of this model similar to the features available for our BP algorithm. The following features were included in the test:

1. **URL:** Included in BP
2. **IP:** Included in BP
3. **Apex domain:** Resembles tokens used in BP
4. **Top level domain:** Resembles tokens used in BP
5. **AS number:** Included in BP

By only using the features listed above and the same parameter settings as described previously, the performance was slightly affected. Leading to an accuracy of 93%, with the features used being more similar to the ones later used by the BP solution.

6.2 BP Results

As a starting point, our graph consisted of three different node types: domains, tokens, and IPs. The following results show how the accuracy of our model improves as more nodes and edges are added. The first results shown make use of the Epsilon model that was described in the method section. The graphs with the best accuracy and percentage of predictions will be subject to the other models, Word2Vec and Node2Vec.

1. **Graph 1 (G1):** Contains three node types: domains, tokens, and IP addresses. This graph functions as our most basic approach.
2. **Graph 2 (G2):** G2 is built upon G1 with the addition of CIDR blocks for IP addresses, connecting each IP to its corresponding CIDR block. This allows risk to propagate further and increases the connectivity of the graph.
3. **Graph 3 (G3):** G3 is constructed similarly to G2, but instead of using CIDR blocks to connect IP addresses this graph uses the IP's ASN.
4. **Graph 4 (G4):** G4 is built upon G1, the addition to this graph is the inclusion of Levenshtein neighbors. This addition connects domains to other domains with similar URL strings.
5. **Graph 5 (G5):** G5 is a combination of G2 and G3, the difference here is that the CIDR blocks will resolve to an ASN in the order IP-CIDR-ASN.
6. **Graph 6 (G6):** G6 is a combination of G5 and G4, meaning it uses the relation IP-CIDR-ASN, as well as the Levenshtein edge domain-domain.

In Table 6.2, we present some of our key results. The highest accuracy, as well as the highest percentage of predictions, are in bold. What is important to consider here is that "Acc1", is the accuracy of the model when including domains that did not receive a prediction as an incorrect classification. Moreover, "Acc2" is the accuracy when excluding the domains that did not receive a prediction. A domain doesn't always receive a prediction since it may not be connected to a component that is capable of spreading the risk. Furthermore, a domain does not receive a prediction if its final belief is 0.5 malicious and 0.5 benign.
Given these results, we went on to further evaluate G5 and G6 and replace the Epsilon model with Word2Vec and Node2Vec. The results for G5 is presented in

Table 6.2: Evaluation Results for Different Graphs using the Epsilon Model

Graph	False Malicious	False Benign	Predicted (%)	Acc1	Acc2
G1	403	2201	86.46	0.78	0.90
G2	363	2546	91.01	0.81	0.89
G3	233	3130	98.53	0.87	0.89
G4	1880	5955	88.64	0.62	0.70
G5	277	2997	98.60	**0.88**	0.89
G6	1328	6672	**98.87**	0.72	0.73

Table 6.3 and the results for G6 is shown in Table 6.4

Table 6.3: Evaluation Results for G5 using three different Models

Model	False Malicious	False Benign	Predicted (%)	Acc1	Acc2
Epsilon	277	2997	98.60	**0.88**	0.89
Word2Vec	306	3207	97.65	0.86	0.88
Node2Vec	296	3039	**98.62**	0.87	0.89

Table 6.4: Evaluation Results for G6 using three different Models

Model	False Malicious	False Benign	Predicted (%)	Acc1	Acc2
Epsilon	1328	6672	98.87	0.72	0.73
Word2Vec	312	3023	98.08	0.87	0.89
Node2Vec	280	2922	**98.89**	**0.88**	0.89

6.2.1 LSTM model

To be able to consistently predict all domains, we incorporated an LSTM model to push the initial potentials of the unlabeled domains, as was described in the method section. The model was trained with a sub-sample of domains, 20 000 benign domains and 20 000 malicious domains. Both the accuracy and loss when training the LSTM model are shown in Figure 6.4, where the blue lines represent the training data, while the orange lines represent the validation data. The training was carried out for ten epochs and the loss remained quite high. The validation accuracy starts and ends at about 90%, however, when evaluated against the entire set of URLs the accuracy equals approximately 92%. The figure also shows that the validation accuracy is consistently slightly higher than the training accuracy. This is most likely caused by random variability and the fact that the domains are quite similar

to one another. However, for future work, this would require further evaluation as there could be other issues causing this, such as overfitting or data leakage.

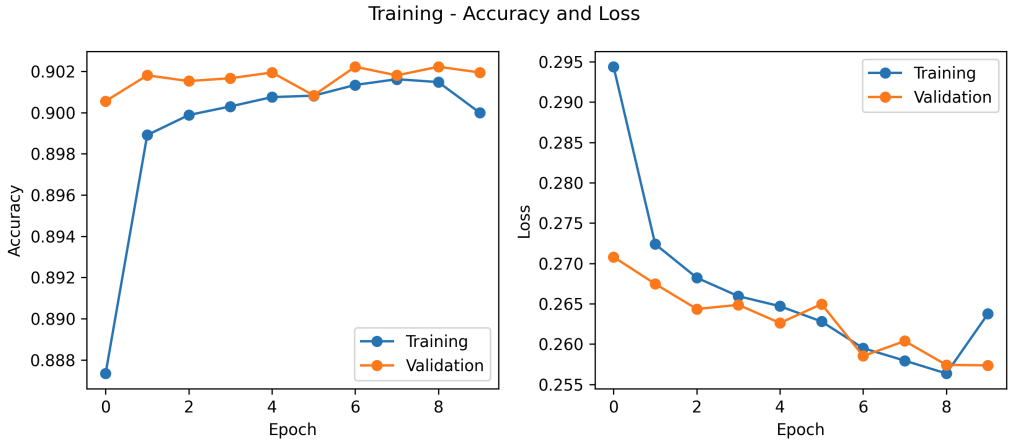


Figure 6.4: Results from training the LSTM model for ten epochs, where the blue lines represent the results from the training set, while the orange represents the validation set. The left figure displays the accuracy and the right figure displays the loss.

In the following results, the prediction rate is 100% and will therefore be excluded from the tables. Furthermore, since "Acc1" and "Acc2" are now equal, only "Acc1" is presented. The results for graph G5 are presented in Table 6.5 and the results for G6 can be seen in Table 6.6.

Table 6.5: Evaluation Results for G5 using three different Models assisted by an LSTM model

Model	False Malicious	False Benign	Accuracy
Epsilon	18	3033	0.90
Word2Vec	14	2882	0.90
Node2Vec	16	2921	0.90

Table 6.6: Evaluation Results for G6 using three different Models assisted by an LSTM model

Model	False Malicious	False Benign	Accuracy
Epsilon	576	6099	0.78
Word2Vec	14	2834	0.90
Node2Vec	13	2853	0.90

6.3 Final model

Studying the results shown in Table 6.5 and Table 6.6 the performance is very similar between graph constructions. The added URL model also had a small impact on the overall accuracy, when comparing the results with Table 6.3 and Table 6.4. The final model was selected based on accuracy, prediction rate, and false positive rate. The selected model, therefore, ended up as the Node2Vec solution with the LSTM model, using the G6 graph. An illustration of the final model can be seen in Figure 6.5. Where "Build graph" corresponds to constructing the G6 graph, and the BP algorithm uses Node2Vec and the URL model as it is initialized.

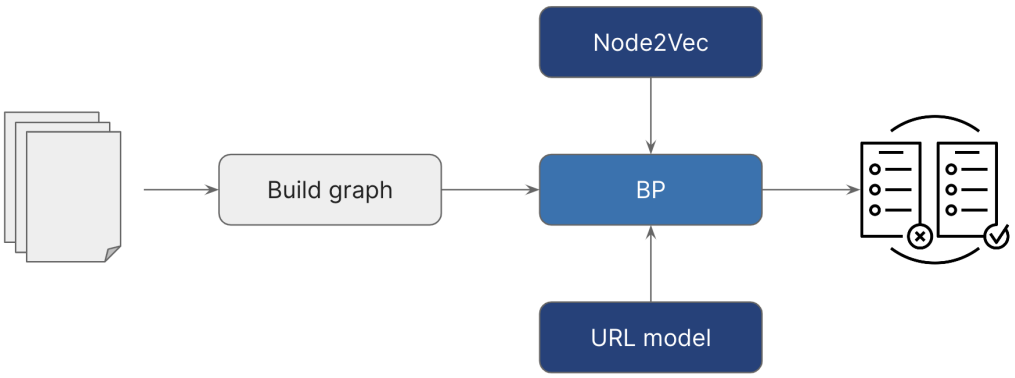


Figure 6.5: Illustration of the selected final model. The data is used to construct the G6 graph, the BP algorithm then takes advantage of the Node2Vec embeddings and URL model predictions to predict the unlabeled domains.

The model demonstrates good abilities to differentiate between benign and malicious data samples as can be seen in Figure 6.6. The figure shows the ROC curve and the AUC. The AUC score obtained by the model is 0.96.

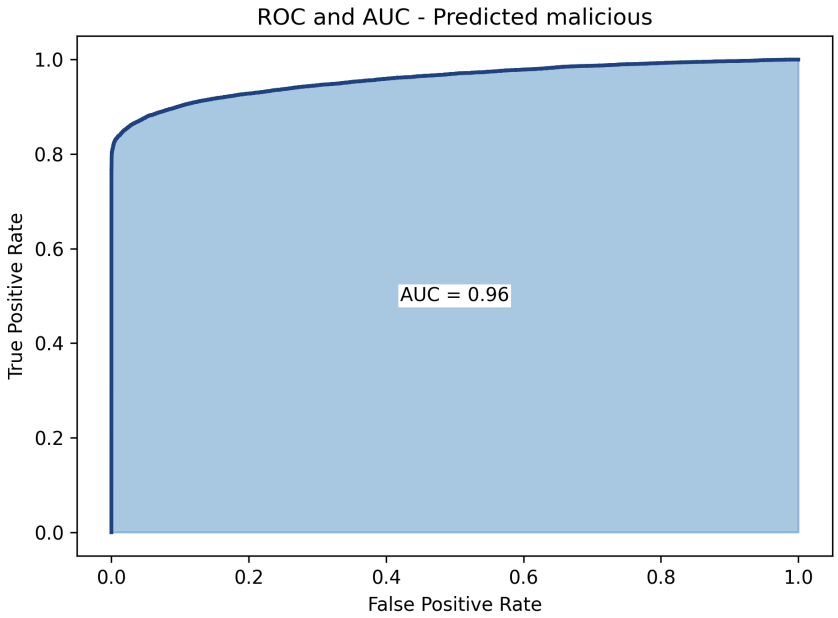


Figure 6.6: ROC curve for the final model with true positive rate on the y-axis and false positive rate on the x-axis. The AUC score is 0.96 which indicates well separated classes.

The distribution of predictions can be seen in Figure 6.7, where the first figure shows the domains predicted benign and the second figure the domains predicted to be malicious. The red graphs display the domains wrongly predicted while the blue graphs represent the domains accurately predicted. The figures display how the model manages to differentiate between the falsely predicted domains, especially towards malicious predictions where the overlap is very limited.

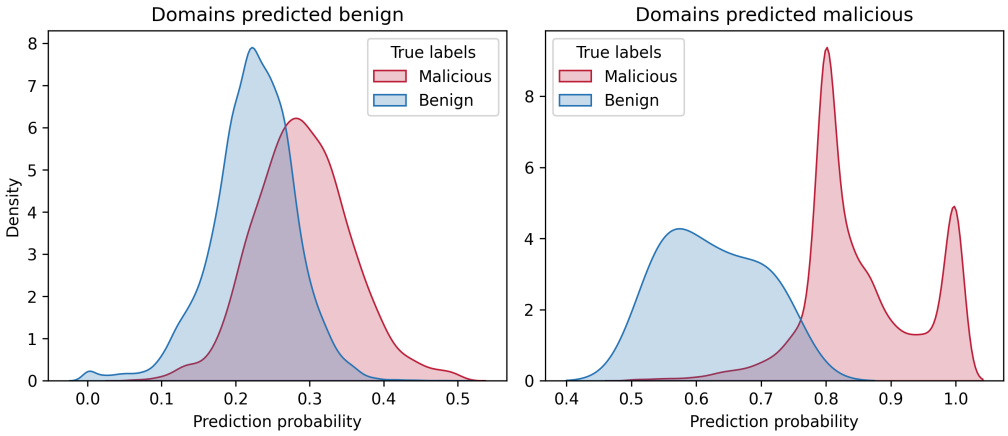


Figure 6.7: Predictions for domains predicted benign, left, and domains predicted malicious, right. The x-axis corresponds to prediction belief and the y-axis density of samples. The red graphs are wrongly predicted samples while the blue graphs represent accurate predictions.

7
Discussion

7.1 Dataset

The dataset that was used for this project may cause misleading results for our RFC model. As stated in the related work section, these models tend to generalize poorly. This is most likely a consequence of the fact that we are using domains that are really benign as well as domains that are really malicious, and nothing in between. This may cause issues for real-time classification since websites such as blogs or restaurant web pages may seem malicious to the RFC model. The reason for this is that small businesses may go for cheaper registration options while still wanting their website to look legit, similar to the behavior of hackers.

The dataset naturally affects the BP as well, perhaps more so due to its size. Our method of BP relies on the fact that every domain's neighborhood is included in the graph, which is usually not the case. Secondly, the dataset consists of more benign domains than malicious, and as can be seen in the BP results, there are a lot more domains that are labeled false benign than false malicious. We believe that in order for BP to reach its full potential, a larger graph is required. It is also important to balance between benign and malicious domains, otherwise one label will most likely have stronger propagation and diminish the other.

7.2 RFC

Based on our results, we believe that RFC models can definitely be a useful tool in detecting phishing domains, especially since they are easy to implement and directly use to classify new domains. The biggest flaw of this model is that its result heavily depends on its dataset. We believe that a good approach for this model is to spend time on the dataset that the model will be trained on. Using domains that are from Tranco and Phishtank gives decent results, but in order to generalize better we believe that it requires a more diverse dataset, including some low-traffic domains such as blogs, restaurant websites, family websites, etc. Having a more diverse dataset would help reduce bias, and make the model more robust for real-world applications.

The promising results indicate that the RFC model manages to find good feature splits and separate benign from malicious samples. Even when only using the features similar to the ones used by BP the accuracy remains at 93%, where the most

important features from previous models were removed.

7.3 Belief propagation

The results of our BP definitely shows potential in classifying domains. The difficulty in this model lies in how to properly structure the graph and propagating risk via IP and tokens is not always correct. Many IP addresses can for example host hundreds of thousands of domains and contain just a few phishing domains, whereas the majority of the domains are unknown. In situations like this, the IP doesn't necessarily tell us that much and propagating risk with a fixed edge potential may cause issues. By extending the standard BP with models like Word2Vec and Node2vec, this type of false propagation can be mitigated. As of now, these models were only used to weaken the connection between two nodes, however, one could also argue that if the link is too weak it should be removed altogether.

Another difficulty lies in how to use models like this in order to classify new domains. One could either build an entire graph based on that one domain, and then classify it, but this doesn't guarantee a prediction. Otherwise, one could construct a large graph, with hundreds of millions of domains, and each time a new domain needs to be classified it can simply be added to the graph. Moreover, this model is not tested in the same way as an RFC model. The graph and all its parameters were tuned in order to give good results, but to really see if the results are general, one should probably divide the dataset into a couple of smaller datasets, to see that the results hold for all graphs and not just a fine-tuned one. Finally, to guarantee some sort of prediction of all domains, we believe this model performs best when combined with a second model, such as an LSTM model.

7.3.1 Final model

In the selection phase of the final model, the optimal solution for our objectives was selected. Several approaches have very similar performances, Node2Vec versus Word2Vec, and with or without the URL model. The end selection as described in Section 6.3 showed the highest accuracy while also minimizing the false malicious predictions.

Minimizing the falsely predicted malicious domains is of high importance to reduce an overflow of possibly malicious domains. Mainly to avoid raising suspicions towards benign domains with honest objectives, but also to minimize investigative work when looking into malicious predictions. The predictions shown in Figure 6.7 show that the prediction has a small overlap for malicious predictions, which indicates these number could be further improved by demanding a higher certainty for a domain to be labeled malicious. Overall the false predictions are more centered around 0.5 and an improvement to avoid uncertain prediction could be to label these as unknown until a higher degree of certainty is reached.

One possible problem when adding the URL model to initialize the node potentials

could be a problem of consequential errors due to relying upon an additional model. To evaluate this new data would be required to determine the effects, however, as of now the results indicates a positive impact when adding the LSTM model.

7.4 Comparison of belief propagation and RFC

Based on the results as well as our methods, we believe that each model has its own place in the world of detecting malicious domains. BP will require more work than an RFC model, as well as a solid framework for tests and experiments to find something that works in the wild, in regards to both computational requirements and results. An RFC model, however, is easy to implement and use, making it a more suitable model for smaller projects. When using RFC with only features similar to the ones you can find in our BP implementation, you can see that the results are very similar, both with an accuracy of around 90%.

Moreover, once a model has been trained for RFC, it requires very little computational power to use, whereas the BP may need to continuously host and update a large graph. In terms of scalability, the drawback of the RFC model we used it that is uses many features that are expensive to retrieve, limiting the amount of classifications that can be done for one day. The data used in our BP is easier to get a hold of, but discussing its scalability is difficult as there are several options on how to use the model as a product.

The BP solutions however offer more interpretability into why domains received the prediction they got, due to the prediction only being based upon the neighbouring nodes. As the size of the dataset increases the graph will be harder to interpret, even still studying smaller sub-graphs allows us to understand limited areas. The BP solution is also expected to handle concept drift or new attack patterns in the data as a far smaller sample of features is used.

Both models, as many others, heavily depend on the quality of the dataset, we therefore believe that the top priority for similar projects should be to create a high-quality diverse dataset.

7.5 Ethical

In this area, there are a few ethical aspects to consider. First of all, it may be harmful to a business if its website is incorrectly classified as malicious. Therefore, all public classifications should always maintain a high quality as well as some explanation behind them. Secondly, there are integrity issues that may arise with the data that is collected. Therefore, it is important to respect the laws that regulate public information, such as GDPR. Thankfully, Recorded Future does its very best to ensure that no laws are broken when collecting public data.

8

Conclusion

First and foremost, it is of paramount importance to find or construct high quality data when building a model to detect phishing domains. Secondly, many models have proven successful in this area. With that in mind, we recommend using models with some form of explainability as it can assist developers that are searching for flaws and potential improvements. Thirdly, it is important to fully test the models ability to generalize before building a product out of it. This can for example be done by testing the model on some low traffic domains and analysing the results. Finally, it is important to recognize the flaws of a chosen model, as well as constantly updating it as hackers develop new evasion techniques.

9
Future Work

For future experiments, we would create one more model using the GNN/GCN framework as it creates a solid framework for embedding more features into a graph based model. Furthermore, we would spend more time to create datasets and testing the model graph model over a few different datasets to make sure it is not overfitted. We would also like to evaluate the Levenshtein connection further by testing different thresholds. Furthermore, there is a modified Levenshtein algorithm that can include an operation that swaps the position of two adjacent characters in a string which would also be interesting to implement. Finally, we would also like to have an experiment that tested our models' prediction compared to those that are currently in place at Recorded Future to see how models like these may improve existing classifications.

Bibliography

[1] Violino. (2023) Phishing attacks are increasing and getting more sophisticated. here's how to avoid them. [Online]. Available: https://www.cnbc.com/2023/01/07/phishing-attacks-are-increasing-and-getting-more-sophisticated.html

[2] Q. Wang, L. Li, B. Jiang, Z. Lu, J. Liu, and S. Jian, "Malicious domain detection based on k-means and smote," in *Computational Science – ICCS 2020*, V. V. Krzhizhanovskaya, G. Závodszky, M. H. Lees, J. J. Dongarra, P. M. A. Sloot, S. Brissos, and J. Teixeira, Eds. Cham: Springer International Publishing, 2020, pp. 468–481.

[3] Najafi *et al.*, "Malrank: A measure of maliciousness in siem-based knowledge graphs," in *Proceedings of the 35th Annual Computer Security Applications Conference*, ser. ACSAC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 417–429. [Online]. Available: https://doi.org/10.1145/3359789.3359791

[4] Wired. (2023) Brace yourself for a tidal wave of chatgpt email scams. [Online]. Available: https://www.wired.com/story/large-language-model-phishing-scams

[5] EC-Council. (2022) The cyber kill chain: The seven steps of a cyberattack. [Online]. Available: https://www.eccouncil.org/cybersecurity-exchange/threat-intelligence/cyber-kill-chain-seven-steps-cyberattack

[6] Bilge *et al.*, "Exposure: A passive dns analysis service to detect and report malicious domains," *ACM Trans. Inf. Syst. Secur.*, vol. 16, no. 4, apr 2014. [Online]. Available: https://doi.org/10.1145/2584679

[7] A. Aljofey, Q. Jiang, A. Rasool, H. Chen, W. Liu, Q. Qu, and Y. Wang, "An effective detection approach for phishing websites using url and html features," *Scientific Reports*, vol. 12, no. 1, p. 8842, May 2022. [Online]. Available: https://doi.org/10.1038/s41598-022-10841-5

[8] P. Maneriker, J. W. Stokes, E. G. Lazo, D. Carutasu, F. Tajaddodianfar, and A. Gururajan, "Urltran: Improving phishing url detection using transformers," in *MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM)*, ser. MILCOM 2021 - 2021 IEEE Military Communications Conference (MILCOM), 2021, pp. 197–204. [Online]. Available: https://doi.org/10.1109/MILCOM52596.2021.9653028

[9] A. J. Ferrante, "The impact of gdpr on whois: Implications for businesses facing cybercrime," 2018.

[10] Kaspersky. What is an ssl certificate – definition and explanation. [Online]. Available: https://www.kaspersky.com/resource-center/definitions/what-is-a-ssl-certificate

[11] L. Tang and Q. Mahmoud, "A survey of machine learning-based solutions for phishing website detection," *Machine Learning and Knowledge Extraction*, vol. 3, pp. 672–694, 8 2021. [Online]. Available: 10.3390/make3030034

[12] S. A. Khan, W. Khan, and A. Hussain, "Phishing attacks and websites classification using machine learning and multiple datasets (a comparative analysis)," in *Intelligent Computing Methodologies*. Cham, Switzerland: Springer International Publishing, 2020, pp. 301–313. [Online]. Available: https://arxiv.org/pdf/2101.02552.pdf

[13] Y. Tsai, C. Liow, Y. S. Siang, and S.-D. Lin, "Toward more generalized malicious url detection models," 2022. [Online]. Available: https://arxiv.org/pdf/2202.10027.pdf

[14] M. Sánchez-Paniagua, E. FIDALGO, E. Alegre, A.-N. M. Wesam, and V. González-Castro, "Phishing url detection: A real-case scenario through login urls," *IEEE Access*, vol. 10, pp. 42 949–42 960, 2022. [Online]. Available: https://ieeexplore.ieee.org/ielx7/6287639/9668973/09759382.pdf

[15] W. Rweyemamu, T. Lauinger, C. Wilson, W. Robertson, and E. Kirda, "Getting under alexa's umbrella: Infiltration attacks against internet top domain lists," in *Information Security: 22nd International Conference*, sep 2019, p. 255–276. [Online]. Available: https://wkr.io/publication/isc-2019-domains.pdf

[16] V. L. Pochat, T. van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen, "TRANCO: A research-oriented top sites ranking hardened against manipulation," in *Proceedings 2019 Network and Distributed System Security Symposium*, feb 2019. [Online]. Available: https://tranco-list.eu/assets/tranco-ndss19.pdf

[17] T. Kim, N. Park, J. Hong, and S.-W. Kim, "Phishing url detection: A network-based approach robust to evasion," 2022.

[18] A. Joshi, L. Lloyd, P. Westin, and S. Seethapathy, "Using lexical features for malicious URL detection - A machine learning approach," *CoRR*, vol. abs/1910.06277, 2019. [Online]. Available: https://arxiv.org/pdf/1910.06277.pdf

[19] C. Johnson, B. Khadka, R. B. Basnet, and T. Doleck, "Towards detecting and classifying malicious urls using deep learning," *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, vol. 11, no. 4, pp. 31–48, 2020. [Online]. Available: https://isyou.info/jowua/papers/jowua-v11n4-3.pdf

[20] H. Shirazi, B. Bezawada, and I. Ray, "'"kn0w thy doma1n name": Unbiased phishing detection using domain name based features," in *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, ser. SACMAT '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 69–75. [Online]. Available: https://shorturl.at/dGJ28

[21] I. Torroledo, L. D. Camacho, and A. C. Bahnsen, "Hunting malicious tls certificates with deep neural networks," in *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security*. New York, NY, USA: Association for Computing Machinery, 2018, p. 64–73.

[22] L. Bilge, S. Sen, D. Balzarotti, E. Kirda, and C. Kruegel, "Exposure: A passive DNS analysis service to detect and report malicious domains," *ACM*

Transactions on Information and System Security, vol. 16, no. 4, apr 2014. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/2584679

[23] P. Lison and V. Mavroeidis, "Neural reputation models learned from passive DNS data," in *2017 IEEE International Conference on Big Data*, 2017, pp. 3662–3671. [Online]. Available: http://home.nr.no/~plison/pdfs/security/bigdata2017.pdf

[24] A. Morichetta, E. Bocchi, H. Metwalley, and M. Mellia, "Clue: Clustering for mining web urls," in *2016 28th International Teletraffic Congress (ITC 28)*, vol. 1, 2016, pp. 286–294. [Online]. Available: https://core.ac.uk/download/pdf/84250919.pdf

[25] J. Prins, "Proactive recognition of domain abuse," 2021. [Online]. Available: https://essay.utwente.nl/84073/1/proactive_recognition_of_domain_abuse_erratum_final.pdf

[26] S. N. Bannur, L. K. Saul, and S. Savage, "Judging a site by its content: Learning the textual, structural, and visual features of malicious web pages," in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*, 2011, p. 1–10. [Online]. Available: https://cseweb.ucsd.edu//~savage/papers/AISec11.pdf

[27] A. Chaiban, D. Sovilj, H. Soliman, G. Salmon, and X. Lin, "Investigating the influence of feature sources for malicious website detection," *Applied Sciences*, vol. 12, no. 6, 2022. [Online]. Available: https://www.mdpi.com/2076-3417/12/6/2806/pdf?version=1646820878

[28] K. L. Chiew, J. Choo, S. Sze, and K. Yong, "Leverage website favicon to detect phishing websites," *Security and Communication Networks*, vol. 2018, pp. 1–11, 03 2018. [Online]. Available: https://pdfs.semanticscholar.org/7d74/3de734311663d9009f361c65d76672654e49.pdf

[29] S. Marchal, G. Armano, T. Gröndahl, K. Saari, N. Singh, and N. Asokan, "Off-the-hook: An efficient and usable client-side phishing prevention application," *IEEE Transactions on Computers*, vol. 66, no. 10, pp. 1717–1733, 2017. [Online]. Available: https://aaltodoc.aalto.fi/bitstream/handle/123456789/28058/A1_marchal__samuel_2017.pdf?sequence=1&isAllowed=y

[30] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1988. [Online]. Available: https://dl.acm.org/doi/pdf/10.5555/534975

[31] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009. [Online]. Available: http://mcb111.org/w06/KollerFriedman.pdf

[32] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, pp. 93–106, sep 2008. [Online]. Available: https://ojs.aaai.org/index.php/aimagazine/article/view/2157/2022

[33] D. Koutra, T.-Y. Ke, U. Kang, D. H. Chau, H.-K. K. Pao, and C. Faloutsos, "Unifying guilt-by-association approaches: Theorems and fast algorithms," in *Machine Learning and Knowledge Discovery in Databases*. Berlin, Heidelberg: Springer, 2011, pp. 245–260. [Online]. Available: https://faculty.cc.gatech.edu/~dchau/papers/11-pkdd-fabp.pdf

[34] P. Najafi, A. Muhle, W. Punter, F. Cheng, and C. Meinel, "Malrank: A measure of maliciousness in siem-based knowledge graphs," in *Proceedings of the 35th Annual Computer Security Applications Conference*. Association for Computing Machinery, 2019, p. 417–429. [Online]. Available: https://dl.acm.org/doi/pdf/10.1145/3359789.3359791

[35] D. Eswaran, S. Günnemann, and C. Faloutsos, "The power of certainty: A dirichlet-multinomial model for belief propagation," in *Proceedings of the 2017 SIAM International Conference on Data Mining*, April 2017, pp. 144–152. [Online]. Available: http://www.cs.cmu.edu/afs/cs.cmu.edu/user/deswaran/www/papers/sdm17-netconf.pdf

[36] ——, *The Power of Certainty: A Dirichlet-Multinomial Model for Belief Propagation*, pp. 144–152. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781611974973.17

[37] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 14, no. 3, 2020. [Online]. Available: https://www.cs.mcgill.ca/~wlh/grl_book/files/GRL_Book.pdf

[38] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations*, apr 2017. [Online]. Available: https://arxiv.org/pdf/1609.02907.pdf

[39] J. Jia, C. Baykal, V. K. Potluru, and A. R. Benson, "Graph belief propagation networks," 2021.

[40] Z. Liu, S. Li, Y. Zhang, X. Yun, and C. Peng, "Ringer: Systematic mining of malicious domains by dynamic graph convolutional network." Berlin, Heidelberg: Springer-Verlag, 2020, p. 379–398. [Online]. Available: https://link.springer.com/content/pdf/10.1007/978-3-030-50420-5_28.pdf

[41] S. B. Kotsiantis, "Decision trees: a recent overview," *Artificial Intelligence Review*, vol. 39, no. 4, pp. 261–283, Apr 2013. [Online]. Available: https://doi.org/10.1007/s10462-011-9272-4

[42] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[43] D. Koutra, T.-Y. Ke, U. Kang, D. H. P. Chau, H.-K. K. Pao, and C. Faloutsos, "Unifying guilt-by-association approaches: Theorems and fast algorithms," in *Machine Learning and Knowledge Discovery in Databases*, D. Gunopulos, T. Hofmann, D. Malerba, and M. Vazirgiannis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 245–260.

[44] Khalil *et al.*, "A domain is only as good as its buddies: Detecting stealthy malicious domains via graph inference," in *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, ser. CODASPY '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 330–341. [Online]. Available: https://doi.org/10.1145/3176258.3176329

[45] L. Yujian and L. Bo, "A normalized levenshtein distance metric," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1091–1095, 2007.

[46] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *CoRR*, vol. abs/1508.07909, 2015. [Online].

Available: http://arxiv.org/abs/1508.07909

[47] W. Ling, C. Dyer, A. W. Black, and I. Trancoso, "Two/too simple adaptations of word2vec for syntax problems," in *Proceedings of the 2015 conference of the North American chapter of the association for computational linguistics: human language technologies*, 2015, pp. 1299–1304.

[48] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013.

[49] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," 2016.

[50] A. Sherstinsky, "Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network," *Physica D: Nonlinear Phenomena*, vol. 404, p. 132306, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167278919305974

[51] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[52] A. Graves, S. Fernández, and J. Schmidhuber, "Bidirectional lstm networks for improved phoneme classification and recognition," in *Artificial Neural Networks: Formal Models and Their Applications – ICANN 2005*, W. Duch, J. Kacprzyk, E. Oja, and S. Zadrożny, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 799–804.

[53] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik, "Gene selection for cancer classification using support vector machines," *Machine Learning*, vol. 46, no. 1, pp. 389–422, Jan 2002. [Online]. Available: https://doi.org/10.1023/A:1012487302797

[54] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, p. 226–231.

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY