DNS2Vec for malicious domain detection

A Novel Approach to Detecting newly registered Malicious Domains through Word Embeddings

Cybersecurity Master Thesis

Nathan Deridder

Delft University of Technology

TU Delft

DNS2Vec for malicious domain detection

A Novel Approach to Detecting newly registered Malicious Domains through Word Embeddings

by

Nathan Deridder

Student Name	Student Number
Nathan Deridder	5839777

Instructor:	G. Smaragdakis
Teaching Assistant:	G. Moura
Project Duration:	February, 2024 - December, 2024
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

Cover:	DALL-E
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Abstract

This thesis explores a novel approach to detecting maliciously registered domain names by embedding these domains. The approach consists of two main steps: embedding each domain, and then classifying it. This method minimizes the need for extensive feature engineering and reduces the amount of data while leveraging the embedder's ability to extract relations between domains. The thesis investigates different embedding techniques, such as Doc2Vec and Meta-Prod2Vec, and evaluates their effectiveness in representing DNS data. The resulting embeddings are tested with various classifiers, including logistic regression, KNN, and random forest, to determine the optimal combination for detecting malicious domains. Our best classifier achieved a precision of 36% with a recall of 18%, indicating that the use of domain embeddings for classifying newly registered malicious domains is possible. In approximately half of the correctly classified cases, the classifier detects malicious domains before Netcraft, on average 18 hours sooner. While the performance of our approach is not convincing enough to replace existing detection techniques, it could be a valuable addition for any Top Level Domain in order to detect some malicious domains sooner.

Contents

1 Introduction

This chapter describes the motivation behind this thesis, it briefly explains the problems faced when trying to detect malicious domains and introduces the concept of representation learning. It also introduces the thesis' research questions, its contributions and its overall structure.

1.1. Problem Statement

Cybercrime is a serious problem globally. According to the Netherlands official statistical office (CBR), 2.2 million people were victims of some sort of online crime in 2022[40] – 14.8% of the population, a staggering number. Similarly, in the United States, the Federal Bureau of Investigation (FBI) estimates that cybercrime costs 12B USD annually [44], showing phishing as the most common form of cybercrime. In Europe, the European Union Agency for Cybersecurity (ENISA) has also deemed phishing as the most common attack vector granting attackers access to systems in Europe [10].

The mitigation of malicious domain activity is a complex undertaking [27], involving multiple different parties. These include the hosting providers [41] and the DNS [26] providers. Hosting providers provide their clients with the physical infrastructure, such as servers, to host a website. DNS (Domain Name System) is the protocol which allows machines to convert human-readable domain names into IP addresses. DNS providers are entities which facilitate this process. An attacker operating a malicious domain requires both web hosting and DNS hosting to work in order to carry out their operations. This means that any party involved in either of these two areas can stop these attacks.

In this thesis, we propose an approach to detect maliciously registered domains, applicable on the DNS level. More specifically, it makes use of representation learning as an intermediary step before classifying malicious domains. In our case a malicious domain name is any domain that is used for malicious purposes, such as phishing, distributing malware and controlling botnets. Many different mitigation strategies for the DNS level already exist [48, 4, 30, 31, 23, 7, 3, 6], however all of these methods require the researchers to do some form of feature engineering. This means that they have to consider which attributes from DNS data (and potentially outside sources) they should use and how to properly encode them. Our approach on the other hand only uses a limited amount of features in order to detect domains.

The dataset we are using has over 60 features per DNS query, and we only use a small subset of this, exploring variations with 10, 3 and 1 feature per query to detect malicious domains.

There are a multitude of reasons why we chose to explore this method. The first reason being that it significantly reduces the amount of data we need to work with. The ".nl" zone receives around 2-3 billion DNS queries a day, each query having over 60 attributes. Additionally, representation learning is able to capture meaning and relationships between domains and the devices querying them, which hopefully make malicious domains stand out and make them detectable by classifiers. While the same data could theoretically be used directly on these classifiers, this would not yield any results as they are not designed to work with this sort of data.

This work is inspired by a blog post published by SIDN Labs about embedding resolvers and domains [42], with our goal extending the embedding of domains to detect maliciously registered ones. We apply our proposed method to all ".nl" domain names, which are maintained by SIDN [36, 39]. Our proposed method itself, however, is not specific to any particular domain zone, meaning that it can be applied to any Top Level Domain (TLD).

Representation learning is a technique which is able to learn representations in the form of vectors (also called embeddings) from raw data alone. In our case we use it to represent any given domain based on DNS query data. The learned representations of these domains are in the form of vectors, which can then be used as features for classifiers in order to detect malicious domains. This means that our proposed method has two major steps:

1. Use word embeddings to learn embeddings (in the form of a vector) of domains based on which devices query those domains.
2. Use the learned vectors as features for classification algorithms in order to detect malicious domains.

In order to approach our goal of detecting malicious domains based on DNS query data, we form a research question as well as three research sub-questions in Section 1.2.

1.2. Research Questions

The main research question of the thesis is as follows:

- How can we apply a combination of representation learning and classification algorithms to detect newly registered malicious domains?

To answer the main research question, we identify the following sub-questions which are each covered in their respective chapters:

1. How can we represent DNS Query Data in a latent space using representation learning?
2. Which classifiers and parameters lead to the highest average precision?
3. What is the Precision and Recall of the final malicious domain classifier?

1.3. Representation Learning

In this section we briefly explain representation learning as well as how it relates to this thesis. Representation learning is a type of machine learning technique which is able to learn

alternate representations from raw data. A famous example are word embeddings, which are able to learn vector representations of words based on their meaning and relationship with one another.

Representation Learning is useful in many different ways, such as capturing complex relationships between occurrences and allowing for data reduction. In the case of word embeddings, many relationships between words such as gender or time, can be effectively captured in the word's embeddings. Embeddings can be regarded as compact summaries of the meaning of their words and their relationship to others. In our case, it serves two main purposes: the first is data reduction by selecting a limited amount of features from queries, which are transformed into a small vector of consistent size. The second reason is that embeddings are able to effectively capture the meaning of concepts and the relationship between them.

1.3.1. Toy Example

In this subsection we cover a toy example of word embeddings as well as how they can be applied in our use case. We to this in order to give the reader more context on the topic before delving into further chapters, which rely on this understanding.

Word Embeddings

Word embedding methods were designed to capture the meaning of words as vectors by looking at the context in which these words appear. We can represent these words as vectors based on attributes that they have, as can be seen in Table 1.1. We specify the rough size and speed in some space for the following words: "Dog", "Cat", "Elephant", "Car", "Truck" and "Scooter". These attributes together can be seen as a multi-dimensional vector representing the object, which we can plot in a graph, see Figure 1.1.

Word	Size	Speed
Dog	0.5	0.7
Cat	0.3	0.5
Elephant	2.5	0.5
Truck	2.8	1.5
Car	1.5	2
Scooter	0.7	0.8

Table 1.1: Example values for different words with respect to their size and speed

In practice depicting words in just two dimensions, in this case speed and size, omits a lot of important information. If one has no information on the actual meaning of these words, one might assume that a cat, car and scooter are all very similar concepts because they are close together in the vector space.

This is where word embeddings come into play, as they are able to learn representations/vectors of the words in its vocabulary. Words with similar meaning are situated close to each other in this embedding space, while words with very different meaning are located far away.

Word embedding algorithms made their way out of Natural Language Processing and are now used to embed all kinds of different concepts for recommendation systems, from products in stores [15], to music [16, 17], to short-term accommodation rental [14]. Additionally, word embeddings have already been applied to DNS query data in order to embed domains as well

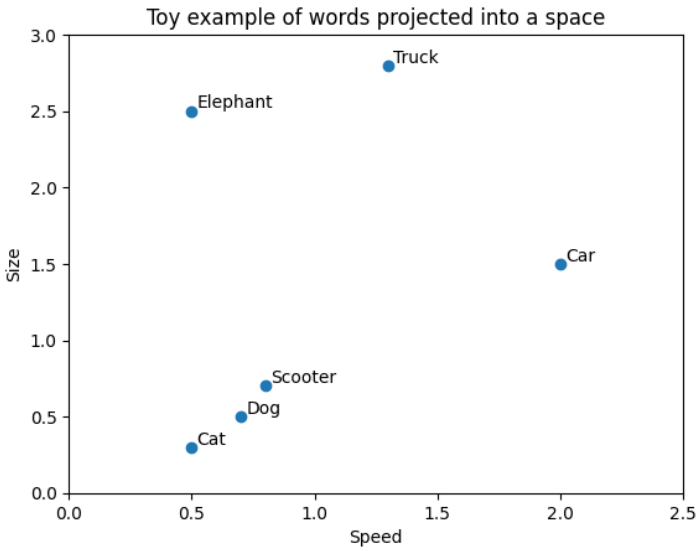


Figure 1.1: Toy example of embedded words

as the devices querying them [42]. This is a good indication that word embeddings could also be a good stepping stone in enabling the detection of malicious domains based on their traffic.

1.4. Methodology

In this section we briefly cover our methodology for detecting maliciously registered domains. A simplified version of the methodology is shown in Figure 1.2.

The first step in our process is collecting raw DNS data from queries sent to authoritative servers, and extracting a varying number of features. These features are used as input to the embedder, which in turn generates an embedding for each domain (Described more thoroughly in Chapter 4).

Now that the domains are embedded, we have a new representation of them that is more compact and has relationships between the different domains. This allows us to use a traditional classifier to make predictions using the embeddings as input.

1.5. Contribution

Our main contribution with this thesis is exploring the feasibility of detecting newly registered malicious domains by embedding domains using their DNS traffic data. More specifically, the goal is to detect malicious domains before Netcraft [28], a service offering a blacklist of domains.

In order to achieve this, we first explore feasible ways to embed domains using DNS query data and choose the most optimal one. We then experiment with different classifiers and hyperparameters to determine the optimal classifier. We examine the performance of our final classifier in a setting identical to a real-world deployment scenario.

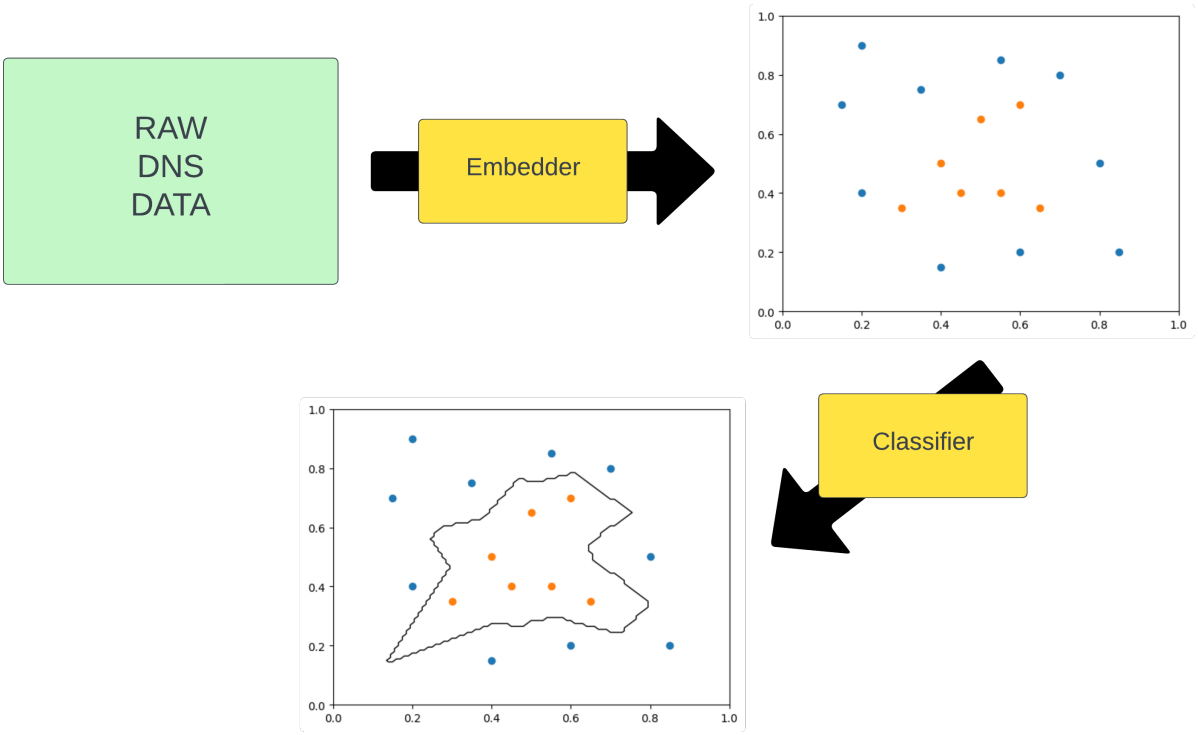


Figure 1.2: Rough Example of Proposed Method

1.6. Thesis Structure

In this report we first cover background information necessary for the readers to understand the thesis, such as what DNS or Representation Learning are.

We later cover related work to this thesis, which includes more advanced embedding methods, as well as the state of the art of malicious domain detection. We explore the topic of malicious domain registration using DNS query traffic more closely.

After these two more informative chapters, we go about answering our three research sub-questions, one chapter per question.

Chapter 4 covers the process of embedding domain names based on DNS Query data. It explores different embedding techniques and determines which would be the best for our use case.

Chapter 5 covers the process of using the embeddings to classify maliciously registered domains, as well as different improvements that can me made. More specifically, it explores the different hyperparameters of the embedder and classifier and determines the best ones.

Using the knowledge gained in Chapter 5, we are able to construct a final classifier in Chapter 6. We test the performance of this classifier using a setup identical to deployment; we use all registration data within a certain time frame, we train the classifier over multiple months, and we separate the training months from the testing months.

After covering the three sub-questions in the chapters, we can answer the main research question in the conclusion, as well as give an explanation of the results achieved during the thesis.

2

Background

2.1. DNS

The Domain Name System (DNS) is a network protocol that enables human-readable domains, such as "tudelft.nl" to be translated into machine-readable IP addresses. This allows us humans to type in a url when navigating to a website instead of having to remember the IP address.

More specifically, if someone queries "tudelft.nl", then the device sends this query to a DNS resolver, which is typically handled by the user's Internet Service Provider (ISP). This resolver recursively sends requests to nameservers until it has received the IP address for the domain the user is looking for. If we consider the domain "tudelft.nl", then the first query that the resolver would send would be to one of the 13 root DNS nameservers. These have information on all of the Top Level Domains (TLD) (e.g. .com, .nl, .org), the organizations in charge of the TLD's are called registries. Now that the resolver has the IP address for the ".nl" authoritative nameservers, it then sends a request to that server for "tudelft.nl", which then references a second level domain. These are primarily managed by the domain's registrars, who are the link between the domain's owner (registrant) and the registry. The nameservers of the registrants can then either return an IP address (meaning that the DNS search was successful), or they can point to other nameservers. The resolver keeps following this trail of references until it gets the domain's IP address (or a DNS resolution failure occurs).

In practice this process is a bit more complicated, since resolvers also store the IP addresses they've requested in their cache. This means that if the resolver now gets another request for "tudelft.nl", instead of going through the entire process again, it already has the domain stored in its cache and can just send it straight back to the user. This is also true for TLDs, so if "sidn.nl" is now requested, the resolver might not have the address of the entire domain saved, however it does still remember the IP address for the ".nl" domain, so it can skip the step of asking for that IP address to the root DNS nameserver.

The key thing to remember for this thesis is that when a domain name is requested by a device, this request is sent to a resolver, which then sends queries to the related nameservers in order to resolve the domain's IP address. Since this method makes use of the vantage point of registries (in this case ".nl"), it is able to see all queries to domains in the zone, as well as

what resolvers they came from. Using this traffic data our method infers information about the domain itself, namely whether it is malicious or not.

2.1.1. SIDN

This master thesis is done in conjunction with an internship at SIDN Labs. SIDN (Stichting Internet Domeinregistratie Nederland) is the entity responsible for registering .nl domain names, as well as making them available on the DNS [36]. SIDN Labs is SIDN's research arm, tasked with further enhancing the security of the internet infrastructure through applied technical research [39].

2.1.2. Malicious Domain Names

Malicious domain names are domains which are used for illegal online activity. Examples of this include phishing websites, Command and Control (C&C) servers for botnets and malware distributors. We distinguish malicious domains into two different categories, maliciously registered and compromised domains. In both cases the domain serves malicious purposes in the present, however the maliciously registered domain was registered by an attacker with the express purpose of executing malicious tasks. A compromised domain on the other hand is a benign domain which is being used for malicious purposes by an attacker. This can happen in many different ways, an example would be an attacker gaining access to the device hosting domain, or the attacker changing the IP address associated with a certain domain.

In this thesis we specifically look at maliciously registered domains because this significantly reduces training complexity. By reducing the problem to detecting maliciously registered domains, we limit the amount of domains that have to be embedded from the total of around 6.2 Million [38] domains, to around 2.000 domains per day. This reduces training times for the embeddings from multiple hours for a few days of data, to multiple hours for an entire year of data.

2.2. Representation Learning

All types of Machine Learning (ML) require some form of representations of data in order to function effectively. These representations, also called features, are measurable properties or attributes from the training data. An example of a feature might be numerical, such as size or speed, as discussed in Subsection 1.3.1, or categorical, like whether it is an animal or a vehicle.

Each feature that is used is supposed to provide the machine learning model with useful information that, when combined with other features, allows it to make accurate predictions.

The traditional way of doing this is feature engineering, where the developer selects features they think might be useful. Extracting good features from data is an important part of Machine Learning, as they are the information the ML model receives in order to make predictions. In machine learning there is the saying "garbage in, garbage out", if the features we select are not useful, then no matter how good the machine learning method is, it will not be able to make good predictions [48]. With feature engineering there is the possibility that important features might be overlooked and not included, resulting in worse results.

This is where representation learning comes in, a technique which automatically transforms raw data into a usable representation of that data. A particular kind of representation learning that we are interested in is word embeddings, as they are able to not only learn new representations of entities (words, items, resolvers) in a certain context, but it is also able to learn and incorporate different relations between the entities, and correctly preserve these from the raw data to the new vector space itself. This makes word embeddings stand out in terms of representation learning techniques, as these relations can now more easily be recognized by other machine learning methods, who can now use these for more complex predictions.

2.3. Word Embeddings

Word Embeddings were originally created for Natural Language Processing (NLP) tasks and is a subcategory of representation learning. The goal of word embeddings is to learn meaningful representations of words (in the form of vectors) based on the word's context. As already briefly explained in Subsection 1.3.1, if the vectors of two words are close together in the vector space, then this implies that the trained embedder considers these words to be similar. Words with very different vectors are supposed to have very different meaning. They learn the meaning of words based on the context they appear in (their relation to other words). The entire method builds upon the assumption that similar words appear in similar contexts. The surrounding words of a particular word are considered as that word's context, so if the context of two different words is similar, then they are put closely together in the vector space.

While in the toy example described in Subsection 1.3.1 had dimensions which are very clear (speed and size), the meaning of the dimensions learned by word embedders is not known. While we don't know what the individual dimensions mean, we do know that they hold some form of significance, proven by a famous example in word embeddings: If you take the embedding for "king", subtract the embedding for "man", add the embedding for "woman", the closest embedding to this result is the embedding for "queen" [25]. This remarkably simple example demonstrates that word embeddings are able to project words into a multi-dimensional space while still maintaining the relations between the different elements.

2.3.1. Word2Vec

Word2Vec was introduced in mikolov et al. and was the basis for modern word embeddings. It was the first widely used word embedding algorithm in Natural Language Processing, and later also in countless other fields. Word2Vec is a self-supervised machine learning algorithm, which means that it does not require any labelled data to train, but rather learns from the contexts in which words appear.

With supervised learning all the training data is labelled, which is a very time intensive endeavor when the labelling is done by hand and the dataset is large. In the method proposed by Mikolov et al., they are able to avoid this manual labelling by using the context (surrounding words) as labels for the target word. The underlying idea behind this is the Distributional Hypothesis [34], which states that words with similar meaning appear in the same contexts, and thus have the same words surrounding it. If we place a word into a vector space based on all the words it's related to (its context), then a word with similar meaning will have similar

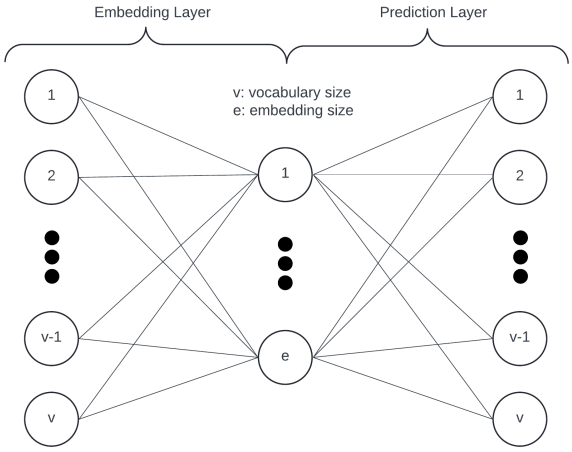


Figure 2.1: Depiction of the Word2Vec Model, which has an embedding and a prediction layer. v is the vocabulary size, e is the embedding/dimension size

Word	Index	Embedding	
King	1	0.4	-0.2
Queen	2	0.9	-0.2
Man	3	0.4	0.6

Table 2.1: Embeddings of example in 2.2

contexts, and thus will also be placed in a similar area in the vector space.

As can be seen in Figure 2.1, the Word2Vec model consists of two fully connected layers, the embedding layer and the prediction layer. The embedding layer is a fully connected layer where the input size is the size of the vocabulary and the output size is the number of desired dimensions for the embeddings. Each word in the vocabulary is assigned an index which corresponds to a node in the input. Since the layer is fully connected, each input node is connected to all the nodes in the Hidden/Projection layer. The embedding of a particular word is simply the weights of the corresponding input node to all the different hidden nodes form.

When considering the simplified example of a trained word2vec model in Figure 2.2, the embeddings of the different words are as shown in Table 2.1. The first layer has three nodes (the size of the vocabulary) with each node inside it corresponding to a specific word. In this case, node 1 represents "king", node 2 represents "queen" and node 3 represents "man". The second layer is the projection layer which is the size of the chosen dimension size of the inferred vectors. The last layer is the prediction layer, which assigns a probability to each word in the vocabulary given the input.

Word2Vec is trained by first getting an entire sentence and selecting one target word. It then collects all words within a certain distance of the target word, called the context words. Both the context words as well as the target words are encoded into one-hot encoded vectors, which are a type of vector where each index refers to a certain category or concept. If the location for a specific category is set to 1, then that means that the category is present, otherwise it is not present. For example if we use a vector which conveys which animals are in

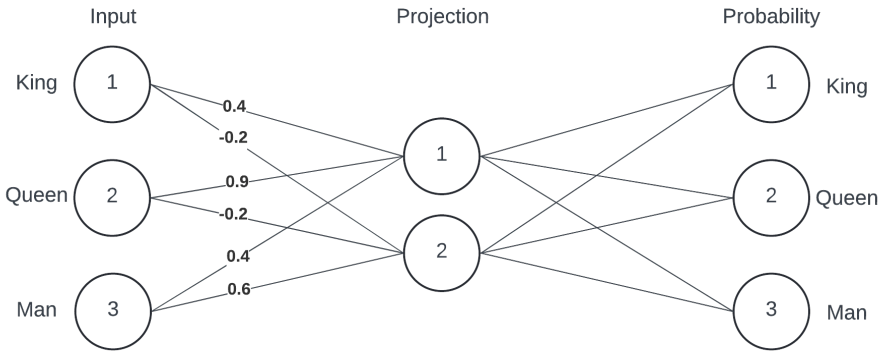


Figure 2.2: Example of Trained Word2Vec Model.

a picture, we might define the positions of the animals in the vector as [dog, cat, bird, horse, cow]. If a picture has a bird in it, then that one-hot encoded vector is [0, 0, 1, 0, 0].

There are two similar ways of training a Word2Vec model, skip-gram and continuous bag-of-words (CBOW). The skip-gram model tries to predict the context words based on the target word. It accomplishes this by taking the one-hot encoded target word as input, and attempts to predict the context words. When the input is passed, the one-hot encoded vector propagates through the embedding layer and the prediction layer, and returns probability of each word being the context word. Backpropagation is then applied, where the weights of the neural network are altered in such a way to get the probabilities for the correct context words as close to 1 as possible, while keeping all other context words as close to 0 as possible. This process of selecting a target word and context words, predicting the context words based on the target word and adjusting the weights accordingly is repeated for each word, and eventually leads to a trained model where each word has a trained embedding.

CBOW works in a similar fashion, except that it predicts the target word based on the context words. This is done by taking the context words and converting them to their current vectors. These vectors are then combined, either by averaging or summing, at which point the prediction layer then again outputs the probability of each word being the target word. Backpropagation is then again applied to tune the weights, and this entire process is applied to all words.

Once training is complete, the model now has an embedding for each word in the vocabulary, which should accurately capture relationships between words. The actual vectors/embeddings are the weights inside the embedding layer, so in the example from Figure 2.2 the embedding for "King" is [0.4, -0.2], "Queen" is [0.9, -0.2] and "Man" is [0.4, 0.6], as can also be seen in Table 2.1.

3

Related Work

In this chapter we cover the related work of this thesis, which is divided into two sections. The first section covers alternative embedding techniques in Section 3.1 which could be applicable to our use-case. The second part of this chapter we focus on previous work done on the detection of malicious domain names.

3.1. Alternative Embedding Techniques

In this section we describe more embedding techniques that go beyond word2vec, such as product embeddings and embeddings with metadata. We describe these as they are viable alternative embedding techniques which could be useful in embedding domain names.

There are many different embedding techniques that exist, however in this section we only cover Prod2Vec and Meta-Prod2Vec as this is what we use for our method. We briefly discuss other embedding methods as well in Subsection 4.2.3, and the reasons why we chose not to use them.

3.1.1. Product Embeddings

As the authors of the Prod2Vec paper [15] realized, Word2Vec is not only useful for mapping words into a space by looking at the context they appear in, but that the same method can also be applied to completely other contexts. The authors propose embedding Yahoo Mail users based on the Products they have purchased to deliver better ads. One important thing to note is the fact that the authors do not propose a new algorithm or method, but rather propose a new use case for an existing method (Word2Vec).

In the case of Prod2Vec, each product represents a word in the Word2Vec algorithm, and sentences are constructed based on which products a user bought together. In other words, products are embedded based on what other products are frequently bought by the same users. The premise for Word2Vec, namely that similar words appear in similar contexts, remains the same for product embeddings. In this case the assumption is that similar products appear in similar contexts (products in a shopping basket).

If we consider an imaginary electronics webshop shown in Table 3.1, the store can embed its items based on the orders of its customers.

Item ID	Price	Manufacturer	Product
25	200-300	AMD	CPU
76	300-400	Intel	CPU
78	400-500	AMD	GPU
52	500-600	Nvidia	GPU
34	50-100	Razer	Keyboard
83	100-200	HP	Monitor
64	100-200	Asus	Motherboard
92	50-100	MSI	Motherboard
3	0-50	Logitech	Mouse
32	50-100	CoolerMaster	PSU
31	50-100	Corsair	RAM
14	100-200	Gskill	RAM
85	50-100	Samsung	SSD
⋮	⋮	⋮	⋮

Table 3.1: Example list of toy web shop

If we then construct sentences based on the product ID in each order, we get sentences similar to the ones shown below:

- Order 1: "Item85 Item52 Item76 Item64 Item14"
- Order 2: "Item85 Item3 Item83 Item34"
- Order 3: "Item78 Item25 Item92 Item31 Item32"
- Order 4: "Item64 Item76"

Each word in the sentences represents the ID of a product shown in Table 3.1. These sentences are then fed to Word2Vec as training data, and it is able to derive information about the products solely based on what other products are frequently bought together. Given enough data, the model is able to accurately place these items in an embedding space, such that similar products (like the same product type) are situated close to one another.

Prod2Vec itself is not a new machine learning technique, it is simply a different use-case for Word2Vec. Prod2Vec uses Word2Vec not for embedding and capturing the meaning of words, but rather embedding and capturing the meaning of products/items, making the recommendation of products trivial. The reason we are interested in this use-case, namely embedding products for recommendations, is because it is very similar to what we want to a achieve. Instead of recommending products which are similar to one another, we "recommend" newly registered malicious domains. The underlying methods are the same, with Word2Vec being shown to not only work in the context of embedding words, but also more abstract concepts such as items. This suggests Word2Vec would also be able to embed things different than items, such as domain names.

3.1.2. Meta-Prod2Vec

This paper expands upon the Prod2Vec idea by including Metadata of products during training of the embedding model [45]. This is done to improve upon the cold start problem as

well as the overall embeddings of the products by giving additional information about the product to the embedding algorithm.

In the paper the authors conducted some experiments to compare the normal Prod2Vec against their Meta-Prod2Vec. They used the 30Music dataset [43] to embed songs based on their ID and Metadata (such as artist information). They found that this did improve performance for cold-start traffic, namely songs that do not appear very often and have inaccurate embeddings because of it.

Meta-Prod2Vec works by adding additional metadata to a sentence of the items. So if we again take the previous example of an electronics webshop, instead of the input to the embedding module being just a list of items, it is now a list of items and their metadata. If we expand the sentences from Subsection 3.1.1, we get the following:

- Order 1: Item85 Price=50-100 Manufacturer=Samsung Product=SSD Item52 Price=500-600 Manufacturer=Nvidia Product=GPU Item76 Price=300-400 Manufacturer=Intel Product=CPU Item64 Price=100-200 Manufacturer=Asus Product=Motherboard Item14 Price=100-200 Manufacturer=Gskill Product=RAM
- Order 2: Item85 Price=50-100 Manufacturer=Samsung Product=SSD Item3 Price=0-50 Manufacturer=Logitech Product=Mouse Item83 Price=100-200 Manufacturer=HP Product=Monitor Item34 Price=50-100 Manufacturer=Razer Product=Keyboard
- Order 3: Item78 Price=400-500 Manufacturer=AMD Product=GPU Item25 Price=200-300 Manufacturer=AMD Product=CPU Item92 Price=50-100 Manufacturer=MSI Product=Motherboard Item31 Price=50-100 Manufacturer=Corsair Product=RAM Item32 Price=50-100 Manufacturer=CoolerMaster Product=PSU
- Order 4: Item64 Price=100-200 Manufacturer=Asus Product=Motherboard Item76 Price=300-400 Manufacturer=Intel Product=CPU

All elements of the same order are still seen as one sentence for training, the metadata inserted into the sentences is treated the same as all other "words"/items in the sentences. The embedding algorithm does not distinguish these, and simply sees them as extra words, which means that the different metadata values are also embedded. As an example, both "Price=50-100" and "Price=100-200" are words which have a corresponding embedding/vector. This serves to give extra context to the items listed in the orders, which helps mitigate the cold-start problem.

Since our goal is to detect newly registered malicious domains, it is expected that the domains have limited amount of information available. In other words, our use-case also faces the cold-start problem, which Meta-Prod2Vec is supposed to improve. For this reason we believe Meta-Prod2Vec to be a viable embedding method.

3.1.3. Document Embeddings

A document embedder such as Doc2Vec [22] has the goal of also training representations of a document, rather only the embeddings of words. A document can be anything from an article to a paragraph or a simple sentence, with the goal of the document embedding being

to capture the overall meaning of the document.

There are two different ways in which Document Embeddings can be trained, with the first one being the simplest, Distributed Bag of Words (DBOW). The goal during training for this approach is to predict randomly sampled words from a document based on the document's ID.

The second technique is the distributed memory model, which is comparable to Word2Vec's CBOW but has the document ID as additional input. It predicts the target word based on the context words as well as the document itself. During training the document ID is used to get the document vector, which is combined alongside the context words. Like with Word2Vec's CBOW, the combined vector is then passed through the prediction layer, which outputs a prediction for the target word, at which point we apply backpropagation. Since the document vector and the context words are combined before the prediction step, both are encoded into the same embedding space.

In our use-case of embedding domain names based on their DNS queries, the domain name can be seen as the document, with the resolvers querying the embedder as the words belonging to the document.

3.2. Malicious Domain Detection

There exists an extensive corpus of methods aimed at detecting malicious domains, each employing unique approaches. Zhauniarovich et al. [49] provide a comprehensive survey of these methods, categorizing them based on the type of data, detection algorithms, and evaluation techniques used.

Antonakakis et al. [2] introduce Notos, a dynamic reputation system that identifies malicious domains using a variety of DNS-based features. These features include statistical data about the network and zones of the IP addresses associated with the domain, as well as lexical characteristics of the domain name itself. Notos models legitimate and malicious domains to generate reputation scores for new domains, achieving a high true positive rate of 96.8% and a low false positive rate of 0.38% in evaluations on a large ISP's network.

Hao et al. [18] propose PREDATOR, a system focused on detecting spam domains at the point of domain registration. By analyzing the bursty nature of domain registrations and the similarity of domains registered together, PREDATOR predicts the maliciousness of new domains. This method, which leverages information about current and past registrations, achieved a recall of 70% with a false positive rate of less than 0.35%.

Choi and Lee [7] present BotGAD, a mechanism for detecting botnets by identifying group activities in DNS traffic. BotGAD leverages the coordinated nature of botnet operations to detect malicious activities in real-time. Using error correction, cluster analysis, and hypothesis testing, BotGAD maintains a detection rate of over 95% with a false positive rate below 0.4%. However, its effectiveness is limited to DNS-based botnets and can be impacted by botnets employing techniques like IP churn. Employing malicious domain detection on the DNS query side makes it much harder for attackers to evade detection, as they cannot influence traffic going towards their domain that doesn't come from them.

FluxBuster is another significant contribution, focusing on detecting malicious flux networks

by passively analyzing large-scale DNS traffic from recursive DNS servers. Over a five-month period, FluxBuster achieved a detection rate of 99.3% with a false positive rate of 0.15%. It effectively identified threats days or weeks before they appeared on public blacklists, although it relies heavily on observed user queries and sufficient IP data collection. The fact that FluxBuster is able to identify clusters of malicious flux networks using DNS traffic suggests that detecting malicious domains based on their resolver traffic is also feasible.

Recent advancements have continued to build upon these foundational works, addressing some of their limitations and introducing new techniques. For example, Mahdavifar et al. [24] present a method that selects features and trains machine learning models using enhanced DNS traffic data. This approach incorporates statistical, lexical, and third-party features, achieving an F1-score and precision of 99.4% on a large dataset. However, obtaining third-party features for newly registered domains remains a challenge.

He et al. [19] developed a graph-based approach utilizing passive A records to construct domain relationship graphs, effectively differentiating between malicious and benign domains. By employing an altered DeepWalk algorithm, they capture relationships within the graph, achieving a recall of 94.3% and a precision of 93.8%. While similar to our proposed approach in capturing domain relationships, their focus on A records contrasts with our emphasis on DNS query traffic. Moreover, their dataset's 50/50 split between malicious and benign domains does not reflect the real-world ratio, posing challenges in generalizing their findings. This paper suggests that using embeddings on DNS data is a valid approach for detecting malicious domains.

Where our method differs from the other approaches is the fact that we apply an Embedder on the DNS query data itself to embed the domains, which are later classified. While many of the previously mentioned papers are similar to our approach, such as the embedding of domains based on the related IP addresses of domains [19], the detection of maliciously registered domains [18, 24], and the detection of malicious domains based on DNS query data [2, 30, 24, 19], we make use of a novel approach to detect malicious domains.

There is one other paper which has a similar approach, namely the detection of malicious C&C domains using word embeddings, which is covered in the following subsection.

3.2.1. Malicious domain detection using word embeddings

Word embeddings do not have to exclusively be used in the context of Natural Language Processing. As Yamada et al. [47] shows, Word2Vec can also be used on DNS queries to help identify malicious domains that belong to a command & control (C&C) server. These servers give commands to devices controlled by malicious software, the DNS requests they get have a specific query pattern to them which Word2Vec is able to extract into its vector representation of the domains. Now that the domains can be projected into an n-dimensional space, they can be compared to other domains. The author's approach to determining whether given domains are malicious is by checking whether the domain is close to any known malicious domain. If it is, then that domain is flagged.

We expand upon this idea, as we do not only consider C&C domains, but any type of malicious domain that is reported. This significantly increases the number of domains that are considered malicious. On top of this we limit our scope to newly registered domains, instead of all possible domains. In our approach we also explore various machine learning methods

and hyperparameters to find the optimal classification model.

4

Embedding DNS Query Data

In this chapter we go over the different embedding methods we have tried out. More specifically, we describe how the DNS data is used for embeddings, what embedding methods are feasible for domain embeddings, and some tests on the performance of these embeddings.

4.1. DNS Data

Before we can do any sort of embedding, we need to know which DNS queries to consider in the first place. Since we consider any queries that were reported within 30 days of registration as malicious, all domain names reported after are considered benign in our use case. One important thing to keep in mind is the fact that the moment a malicious domain is reported its traffic pattern changes. This is because many different parties attempt to connect to this domain for various reasons, such as collecting their data. Since our goal is to detect domains before Netcraft, it is very important to discard any queries that are sent after a malicious domain has been reported, as these might give the embedder and classifer information it would not have in a deployment scenario.

4.1.1. Number of queries

The first thing we need to figure out is how many queries we want to use for the embedding. We do this by selecting a timeframe in which we collect queries to a domain. This timeframe starts at the time of registration, and lasts up until a certain amount of days after the registration.

In order to determine this timeframe we look at the average amount of queries to malicious and benign domains for each day after registration, as can be seen in Figure 4.1. It shows that the amount of queries to maliciously registered domains quickly drops off after just the first day of registration. This can be explained by the fact that many malicious domains are caught within the first few days of registration, as can be seen in Figure 4.2. While the median amount of queries sent to malicious domains is 0 after 5 days, the mean amount of queries is still around 500 queries a day. This shows that a minority of malicious domains are still able to operate at 5 days after registration and beyond.

Although most maliciously registered domain names do not receive any queries after 5 days of registration, we chose to collect all query data up to 10 days after registration. This pro-

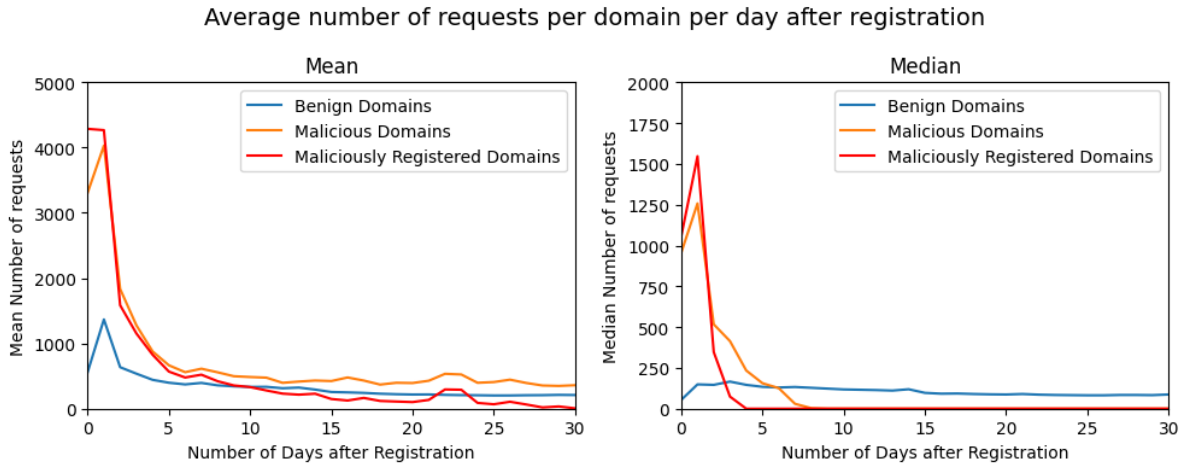


Figure 4.1: Average number of DNS queries to domains per day after registration. For malicious domains, queries are only counted if they are sent before the domain has been reported.

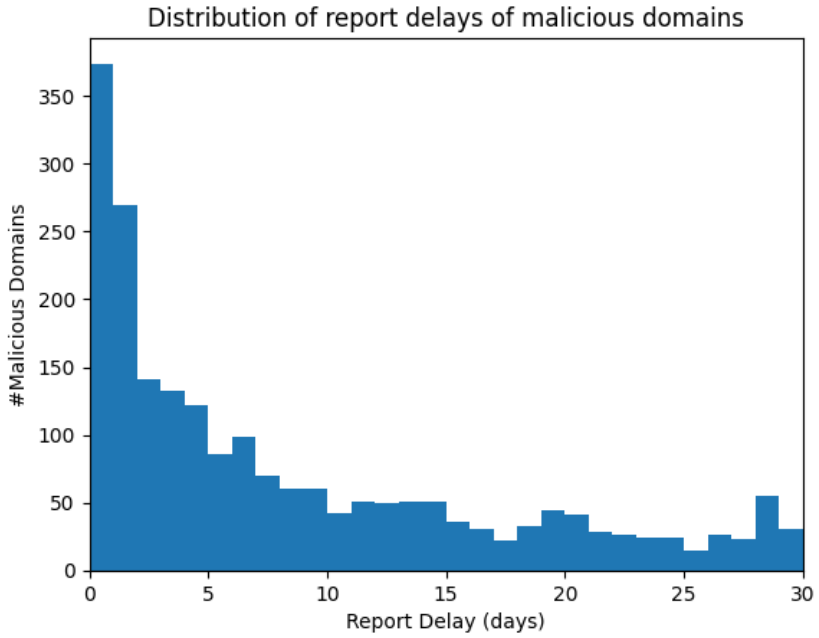


Figure 4.2: Distribution of the report delay (time between registration and report) of malicious domains.

vides a significant buffer for domains which stayed undetected for over 5 days and still receive queries, while still being a manageable amount of data in terms of processing and training. An important thing to note is that this does not mean that the classifier waits for 10 days before making a prediction.

4.2. Embedding Techniques

The first step in implementing our method for detecting malicious domains is by finding a suitable embedding method. There are many different possible embedding algorithms that have been developed over the years [25, 15, 22].

The first embedding algorithm we look at is Word2Vec [25], as this is the first and most

commonly used embedding technique in related works.

4.2.1. Word2Vec

Word2Vec is able to learn representations of words based on the context that those words appear in. It is able to capture relationships between words and project these into a latent space.

If we were to apply Word2Vec to DNS query data, then our words would not be words from a natural language, but rather a resolver's IP address. So instead of a sentence being made up of words, it is instead made up of resolver's IP addresses. We create these sentences based on domains, meaning that for each domainname we collect all the resolvers that have queried that domain. This allows us to embed resolvers based on their querying behaviour.

This approach however only gives us embeddings for words, i.e. the resolvers, what we're actually interested in is embeddings of domains. One possible solution to this is what Spotify did to embed listening sessions based on song embeddings, namely taking the average embedding of all the songs in a listening session [16]. While the goal of Spotify is not directly related to our objectives, the approach is very similar. Spotify trained embeddings of all their songs based on their appearance in user-made playlists. The assumption here is that similar songs are going to be in similar playlists. Using these song embeddings Spotify then took the average of all of them in order to get an approximation of the current listening session, so it can suggest songs that are similar to whatever the user is currently listening to.

One way to test the approach of averaging resolver embeddings to obtain a domain embedding is by comparing their cosine distances. Since embeddings of similar domains should be closer together than embeddings of completely different domains, we can generate two embeddings for the same domain as well as an embedding for a random domain. If this approach works, then the distance between the two embeddings for the same domain should be significantly lower than distance of two embeddings from random domains.

To run this experiment we go over each domain and collect two different sets of resolvers that have queried that specific domain. We then also collect a set of resolvers for a different random domain. We average the embeddings of these three sets to get the 2 embeddings of the same domain and one embedding of the random domain. We calculate the cosine distance between the two embeddings of the same domain, as well as the cosine distance between the embeddings of the 2 different embeddings. This process is repeated for all domains in the dataset, with the distances being plotted in Figure 4.3.

Figure 4.3 depicts the two distributions of the test that we just described. There is a clear difference between the two distributions, which means that the average of resolver embeddings could be a viable way of embedding domains. Despite this, the distributions do not seem very consistent, as they do not resemble a normal distribution at all. Additionally there is a significant overlap between the distribution for the distance of the same domain and the distribution for the distance of the different domain.

This is why we also look into Document Vectors for the embedding of domains, as covered in the next section.

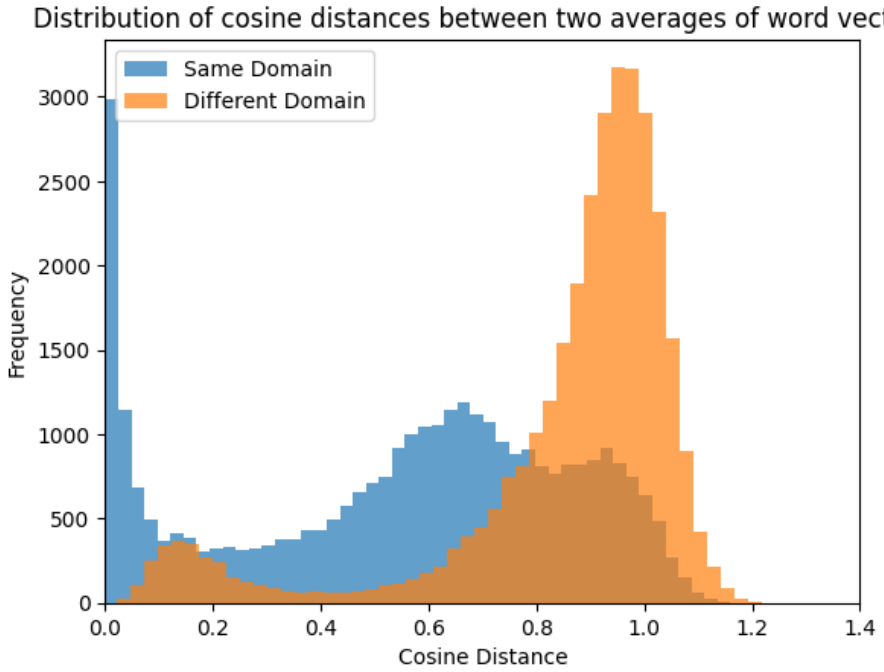


Figure 4.3: Distribution of distances between averaged word vectors for the same and different domains.

4.2.2. Doc2Vec

Instead of averaging the vectors of resolvers to get embeddings of the domains, we can train a Doc2Vec model, as this is able simultaneously train both the resolver and domain embeddings. The Doc2Vec model is able to learn the prevalent characteristics of each document, which can serve as a form of summary. The document embeddings can be trained in a different vector space from the word embeddings, meaning that different features and relations can be extracted and saved about documents.

As is also the case with our Word2Vec approach, in this scenario the resolver IPs that query a specific domain are aggregated into a list/sentence. In the case of Doc2Vec, we also provide a document ID, which in this case is the domain name. This allows the embedder to learn both embeddings of the resolvers as well as the embeddings of the domains.

However simply having embeddings for domains that were trained is not enough for detecting newly registered malicious domains, since our goal is to embed new domains based on queries they've received. Just having trained embeddings of existing domains is in this case insufficient, as we need a way to generate these embeddings for new domains.

Gensim's Doc2Vec has a method which is able to accomplish just this, infer vector, as it can generate embeddings based of unseen documents given any sentence. This means that even if it has never seen a domain before, we just have to pass a list of resolvers that have queried a specific domain and the trained Doc2Vec model generates an embedding of it.

In practice, this allows us to collect queries all queries for a specific domain and put them in a list. Doc2Vec's infer vector method is then applied to this list of resolvers, which gives us an approximate embedding of the domain. This embedding can then be fed to a classifier to determine whether this domain is malicious or not.

Infer Vector Performance

In this section we evaluate whether *infer_vector()* is actually a suitable method for embedding newly registered domains. We do this by running a similar test as in Subsection 4.2.1. Meaning that we train a Doc2Vec model on the DNS query data and get three different embeddings: the trained document embeddings of a domain, an embedding inferred from queries to the same domain, and an embedding inferred from queries to a different domain. We then calculate the distance between the learned domain name embedding and the inferred embedding of that same domain name. Additionally we also take the trained embedding of a domain and calculate its distance to the inferred vector of a different domain. The cosine distance between the trained and inferred embedding for the same domain should be smaller than the cosine distance between trained and inferred embedding of different domains. If this the case, then this implies that *infer_vector()* is able to generate good embeddings.

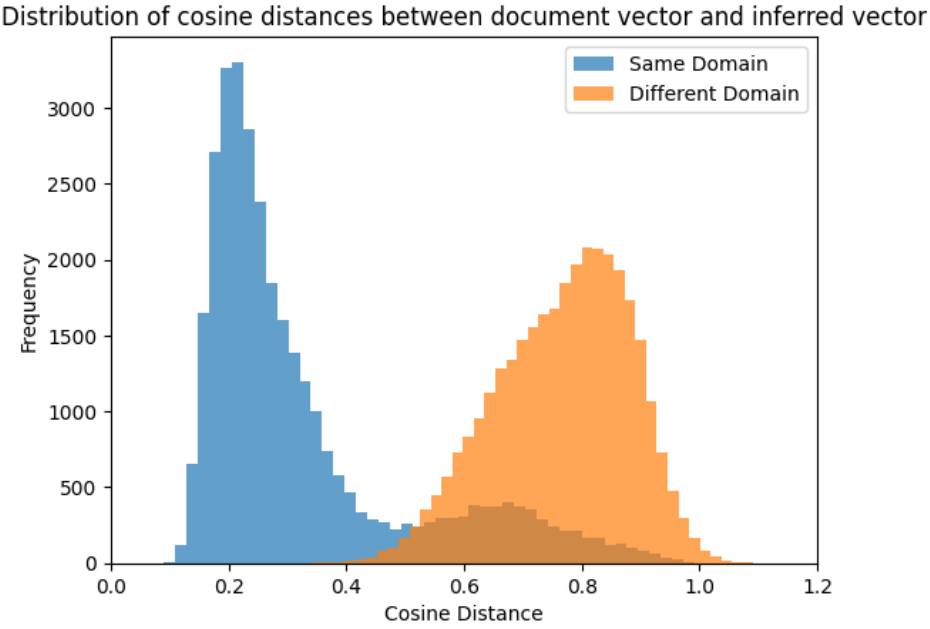


Figure 4.4: Distribution of distances between document and inferred vector for the same and different domains.

The results of this test are shown in Figure 4.4, which shows that there is a clear distinction between the two distributions; the inferred vector of a same domain is much closer to the trained embedding of that domain than the inferred vector of a random domain.

This substantiates that the Doc2Vec *infer_vector()* method works, as it is able to differentiate between inferred vectors of the same and different domains, and it does so more reliably than simply averaging resolver embeddings.

There is one drawback of this approach related to the classifier that will be using the embeddings to make predictions on the maliciousness of the domain names. In machine learning, it is important to get the training scenario as close to the deployment scenario as possible, which is is not yet the case here.

For this reason, we decide to generate inferred vectors for each domain that the classifier will train on, instead of training on the document embeddings. We do this since the prediction

is completed on inferred vectors, so in order to make the training scenario resemble this as much as possible, we also train the classifier on inferred vectors of domains. Another advantage of this that the timeframes of the embeddings are the same. The trained document embeddings are obtained over the entire training data of the embedder (which can span up to 10 days after registration). The classifier however only uses the inferred embedding of a certain amount of queries to make a prediction in order to get the most consistent training and inference performance. By using the inferred vectors of domains as training data for the classifier instead of the document vectors, we ensure that the same method and timeframe are used of the training and testing/inference data, ensuring the training and deployment circumstances are as close as possible.

We verify that this approach of using the inferred vectors of domains as training data for the classifier works by again comparing the distances between embeddings of the same and of different domains. For this test we compare the distances of two inferred vectors of the same domain (generated from two separate sets of resolver IP addresses which have queried the same domain) with the distance of inferred vectors between two different domains.

This ensures that *inferred_vector()* does not simply generate similar vectors for all domains, but that the inferred vectors are domain-specific. We plot the distances between the inferred vectors of the same domaind and the inferred vectors of different domains inFigure 4.5. The figure shows a clear distinction between the distances of inferred vectors for the same domain and inferred vectors of different domains. While the distributions in Figure 4.4 seem better visually, the difference is not enough to warrant using document embeddings to train the classifier, as this cause the training data of the classifier to be significantly different from the production data. It is for this reason that we choose to use the inferred vectors of domains as training data for the classifier.

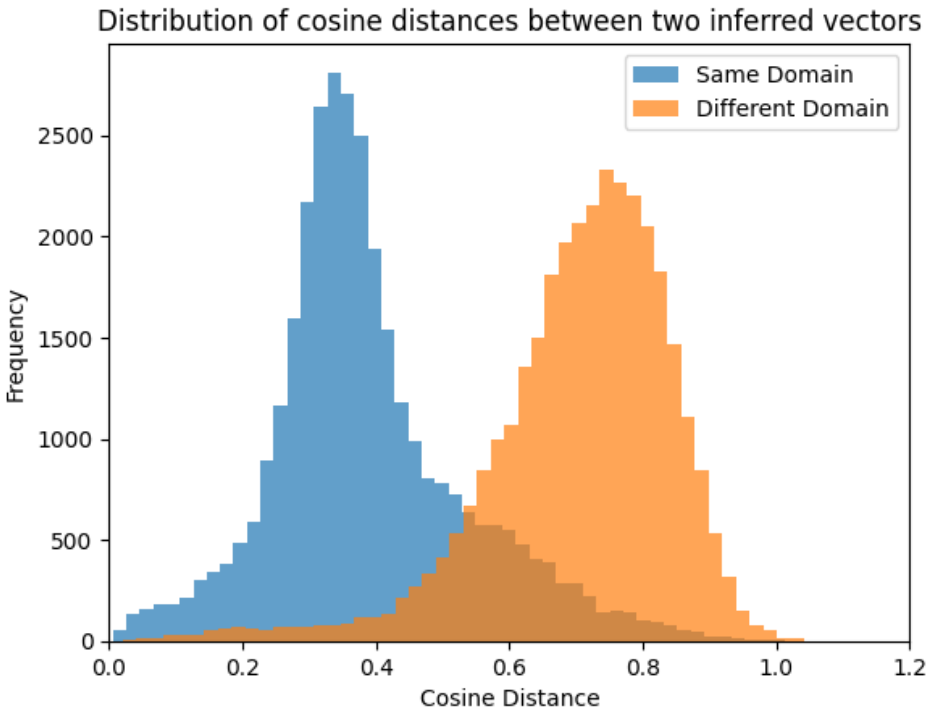


Figure 4.5: Distribution of distances between two inferred vectors of the same and different domains.

4.2.3. Other methods that were considered

Besides the main methods listed above, we also considered other possible ways of embedding domains.

The first option we considered was building a custom implementation of Word2Vec which would train on DNS Query data. We implemented the skip-gram model architecture in Pytorch, however this did not function properly and was also took a considerable amount of time to train, much longer than the gensim implementation, which is why we ultimately decided against using a custom implementation.

Other implementations that we considered, as well as the criteria they meet and do not meet, are shown in Table 4.1.

Embedding Method	Implementation	Maintained	Has Doc Vecs
Word2Vec [25]	Gensim [33, 32]	Yes	No
	Indra [35, 21]	No	
GloVe [29]	Indra [35, 21]	No	No
fastText [5]	Github Repo [12]	Yes	No
Doc2Vec [22]	Gensim [33, 32]	Yes	Yes
StarSpace [46]	Github Repo [11]	No	Yes
flair [1]	Github Repo [13]	Yes	Yes

Table 4.1: Table with various embedding techniques and their benefits and drawbacks

We ultimately choose to use gensim's Doc2Vec, as this is the only embedding method which has a maintained implementation and has document vectors. Flair embeddings [1] also falls under this category, however its main contribution is using character-level embeddings, rather than word embeddings. In our use case of IP addresses, this would be counter-productive, as the individual characters in IP addresses do not have any meaning, it is only the entirety of the IP address which determines its behavior. A single changed number in the IP address can refer to an entirely different device with completely different properties.

4.3. Metadata

We experimented with including metadata about DNS queries in the training of our embedding model, as was introduced by [45] to overcome the cold-start problem that occurs if very little information of a subject is available. When we apply this to our use case, then we would not only use the resolver IP addresses to create sentences, but also other query-specific information. One important step in accomplishing this is determining which metadata should be included, which is covered in the next subsection.

4.3.1. What Metadata?

The database where we get all the query data from is Entrada [37], which stores over 60 different attributes about DNS Queries received by SIDN. Using all of these features as metadata when embedding domains is not feasible, which is why we have to select some features which are likely to help improve the embedding of domains.

We analysed the different attributes that Entrada stores and selected the following as potential candidates for metadata:

- ipv: IP version
- time: unix timestamp
- number of labels: number of labels in a domainname (ex: "ewi.tudelft.nl" has 3)
- qtype: query type
- country: country location of resolver
- asn: autonomous system number of resolver

The query time has to be converted to categorical data, as the embedder cannot work with numerical data. A simple and effective way this can be done is by splitting it up, namely into weekday, hour and minute. This means that the one unix time stamp is converted into 3 metadata points, the weekday, hour and minute. Other metrics such as the day, month and even day could have been included, however we want to keep the amount of metadata low, and assessed that these would likely not contribute much additional information to the embeddings that could help the classifier.

We can now do an analysis of the different metadata attributes to determine whether they could be useful to the embedder and classifier to better detect malicious domains. We download all queries from February 2024 sent up to 10 days after registration of a domain. Using this data we then analyze the above attributes.

Country

We first plot the share of queries to malicious domains per country. We do this by collecting all queries per country, sum up the amount of queries to malicious domains and divide by the number of total queries.

More specifically, we simply calculate the ratio of queries to malicious domains compared to all queries per country, which gives us Figure 4.6.

There are a few countries that stand out, namely Lesotho, Argentina, South Africa, Egypt, Côte d'Ivoire and Nigeria. Lesotho has the highest percentage with 33%, however only 6 queries originated from there, so this comparison is not very fair. On the other hand Argentina and South Africa have 23% and 19%, with a total amount of 42 thousand and 56 thousand queries respectively over the course of a month. This suggests that the country of origin of a query can be a useful indicator in whether a domain is malicious or not, which is why we choose to include it in the metadata for the embedder.

IPV, Labels, QType

Next we compare the distributions of IP version, number of labels, qtype and opcode values for benign and malicious domains. These distributions are further split into buckets of days from 0 to 10 days after registration. We accomplish this by first collecting all queries and splitting them up into buckets depending on how many days after registration they were sent. This is then further split into queries to malicious and benign domains. Then we generate histograms of the values for the IP version, number of labels, query type and operation code. This is shown in Figure 4.7, and if there are any significant differences in the distributions then this means that the attribute could be a helpful indicator in determining whether queries are being sent to a malicious domain or not

However, these attributes do not seem to significantly deviate between benign and malicious domains at any moment in time, which suggests that these metrics are not very likely to

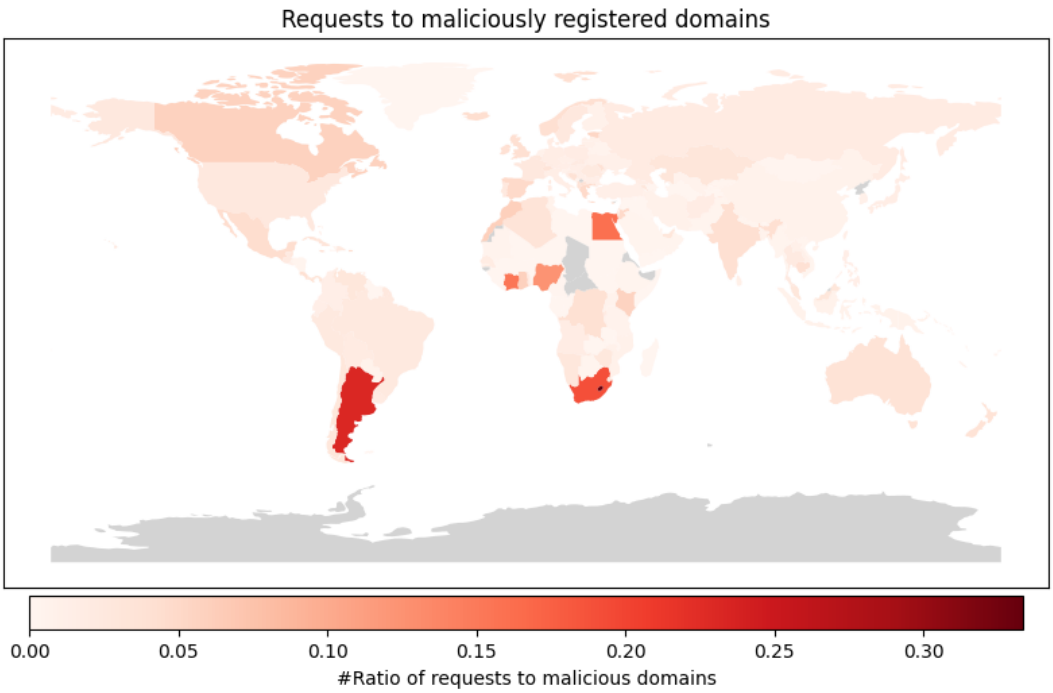
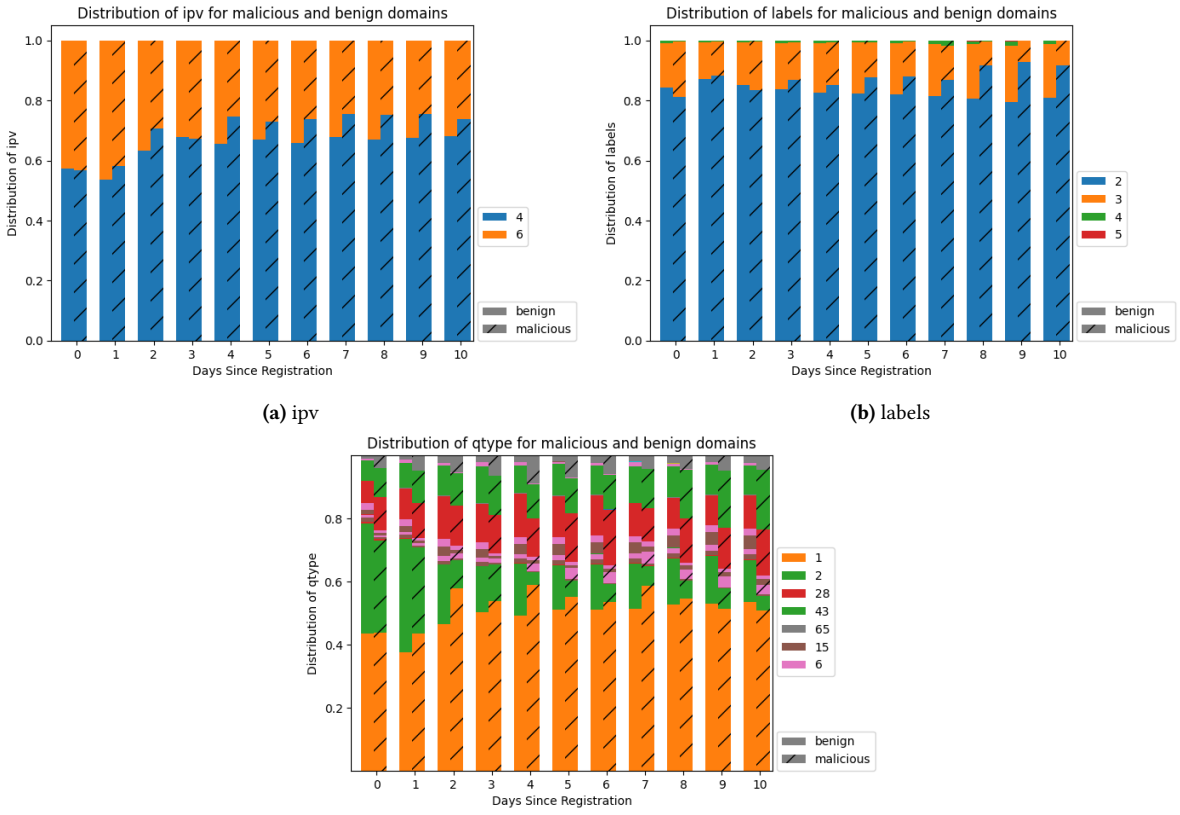


Figure 4.6: Ratio of queries to malicious domains per country.



(a) ipv

(b) labels

(c) qtype

Figure 4.7: Distribution of various metadata attribute values for malicious and benign domains for each day after registration. Values are in percentage

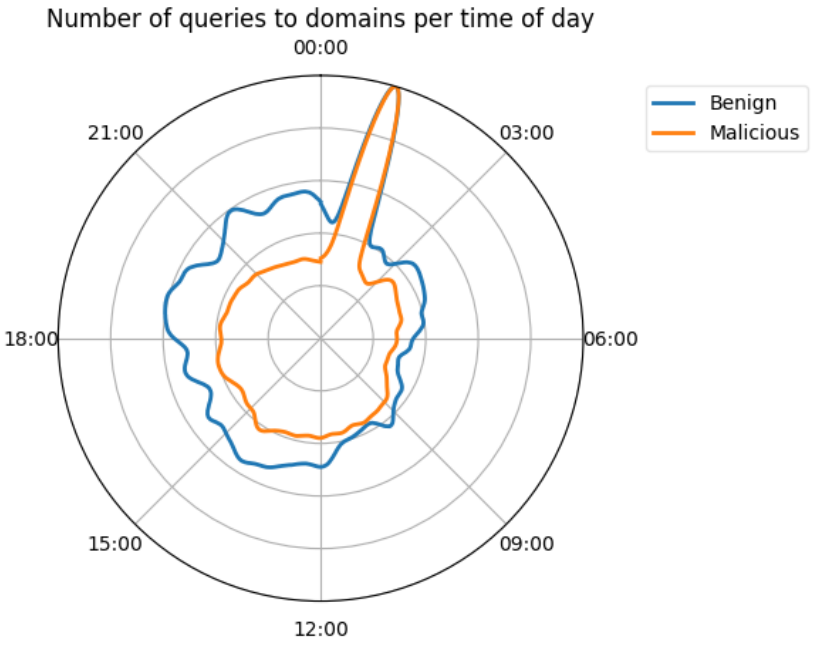


Figure 4.8: Normalized distribution of number of queries per time of day for benign and maliciously registered domains.

provide significant value to the embedder.

Time

Since the embedder cannot work with continuous values, the query's time has to be discretized. An easy way to do this is by splitting up the time into multiple attributes: minute, hour and week day. We visualize whether the time of queries is a good indicator of a domain's maliciousness by plotting the number of queries sent to malicious and benign domains throughout the day. We do this by collecting all queries sent within 10 days after a domain's registration. We then plot the distribution of these times for both benign and malicious domains on a circular histogram, as shown in Figure 4.8.

The plot displays the normalized amount of queries sent to maliciously registered and benign domains per time of day. These values are the average of 11 days of queries after a domain's registration (first day until the 10th included), with a total of 21,740 benign and 33 malicious domains. Both distributions spike around the 1am mark, which is likely related to the fact that the time we use here is converted to CET, would be midnight in UTC in February (when the data was collected). There is likely a lot of automated scanning operations which query domains at the start of every day which cause this large spike. There are not any direct indications that time might be a good attribute when it comes to maliciousness of domains.

ASN

Lastly we consider the Autonomous System Number (ASN) of the resolver as possible metadata for our embedder. Just like the country, more information of the origin of the query might prove useful to the embedder and thus the classifier in determining malicious domains. However there is no good way of displaying and comparing which ASNs are more likely to query malicious domains as the amount of possible Autonomous System Numbers are very large.

Selection

When looking at the different distributions and values of the metadata chosen above, we conclude that the metadata which is most likely to aid the embedder is the resolver's country as well as its ASN. That being said, there is a chance that the other metadata could provide important information to the embedder, information that is not directly perceivable from these visualizations. This is why we choose to test two different metadata configurations when examining the performance of the different hyperparameters in **??**. One configuration only has the country and ASN number, while the second configuration has all the above-mentioned metadata.

5

Classifying Malicious Domain Embeddings

In this chapter we describe and evaluate the classifier which will ultimately detect maliciously registered domains using their embeddings. In order to do this, we first determine which classifiers. Once we have these selected, we run the first experiments using the embedder and classifier. We experiment with different hyperparameters for both the embedder and the classifier, and assess which lead to the best prediction results of the classifier.

5.1. Classifiers

We choose to test on three different classifiers in order to have a balanced picture of the performance of classifying malicious domain names based on their embedding. We choose to test the following three classifiers: K-Nearest Neighbors, Logistic Regression and Random Forest. These three methods are very common and powerful classifying methods.

Random Forest is a collection of decision trees, which at it's core is just a set of many conditions (if/else statements). Decision trees are commonly used in machine learning because of their simplicity, and can be very powerful when combined with multiple other decision trees to form a Random Forest[20].

K-Nearest Neighbors (KNN) is a classification algorithm which classifies new entries according to which class has the nearest k neighbors to the existing "training" data [8]. For example if k is 2 and an entry is given for prediction, then KNN's prediction is whichever class has the closest 2 neighbors. KNN does not need to be trained, the entries and classes just have to be initialized. This makes it a simple and fast classification algorithm.

Logistic Regression [9] is a binary classification algorithm that predicts one of two possible outcomes. It uses the logistic (sigmoid) function to map the input features into a probability between 0 and 1. Logistic Regression is simple, interpretable, and efficient. Unlike more complex models, it is relatively quick to train and does not require extensive computational resources.

Now that we know which classifiers to use, we can determine which hyperparameters we want to test to achieve the best performance. Since we are using an embedder as well as a

classifier to make predictions, we also need to fine-tune two sets of hyperparameters to get the best results.

In order to find the best performing hyperparameters, we make a selection of possible values for each hyperparameter. Combinations of these are then used to train the embedder and the 3 classifiers, where the performance of the classifier being a proxy for the performance of the hyperparameters.

5.1.1. Classifier Evaluation

The most common metrics for evaluating models are the precision and recall. The precision of a model is the amount of correctly classified instances of all the instances marked by the classifier and the recall is the amount of correctly classified instances out of all the actual positive instances. The precision and recall of a model depend on the threshold of the classifier, as the threshold decreases, the precision also increases (because the model becomes less strict on what counts as positive) and the recall increases (because a larger amount of positive instances are flagged by the classifier. More specifically, the precision and recall of a model are inversely correlated, and depend on the chosen threshold.

In order to more accurately represent the actual performance of a machine learning model, there are other metrics which analyze the performance of the model at many different threshold values. One of these is the Receiver-Operator Characteristic (ROC) curve, which measures the True Positive and False Positive rate across many different thresholds. For each threshold, the True Positive and False Positive rates are recorded and plotted, resulting in the ROC curve. This curve can then be summarized by calculating its AUC (Area Under Curve), which results in a number from 0 to 1 which is able to capture the performance of the classifier over its different thresholds.

A drawback of the ROC is that it misrepresents the performance of classifiers on highly imbalanced datasets with very few positive values. This is why our metric of choice is the average precision (AP), which summarizes the precision-recall curve and is still able to accurately depict the performance of a model for highly imbalanced data. For the AP we plot the precision and recall for many different thresholds, which forms a curve. Calculating the auc (area under curve) for the precision-recall curve gives us the average precision, which is again a number between 0 and 1.

For each set of hyperparameter combinations, we generate an embedding model and a classifier using those settings. The classifier is then tested and its AP is recorded. We choose the hyperparameters for the final classifier based on which hyperparameters have the best average precision in this section.

5.2. Testing Hyperparameters

In order to create our training and test dataset, we first need to select which domains to collect queries from, since we are only interested in queries from newly registered domains. For these hyperparameter tests, unless otherwise specified, we train and test on one month worth of query data.

We decide to sample 200.000 benign domains registered over an entire year (May 7 2023 - May 7 2024), as well as use all 2693 malicious domains registered from 2022 onward. We

do not use all registrations within the given time frame as this would extend training and testing times, which is not desirable when we need to train many different models.

More specifically, for all hyperparameter tests in this section, we use queries sent to domain names registered in March 2024, unless otherwise specified. This leaves us with exactly 16,000 benign domains and 135 malicious domains. We apply 5-fold cross-validation, which means that data point (both benign and malicious domains) appears exactly once in the test set. This means that there are 5 different classifiers trained, each having an entirely different test set from the others. We do this to ensure that we have an accurate picture of the classifier's performance across all data.

Once we have the domains we want to train on, we can generate the sentences (sets of queries to a domain) which will be used to train the embedder. We select all queries that fall within 10 days of a domain's registration and belong to domains in our registration dataset mentioned above.

Using these queries we can now generate one or multiple sentences per domain, either with or without metadata. These sentences are then used to train the embedder. For each domain we get the first sentence and use the trained embedder to infer an embedding vector. These vectors represent the embedding of the domain, and are used to train the classifier. More specifically, we split the existing domain names into a train and test dataset, with the embeddings of the domains in the train dataset being used to train the classifier, and the domains in the test dataset being used to test the classifier. Finally, we calculate the classifier's average precision (AP) and store it so the various hyperparameters can be compared.

5.2.1. Embedder Hyperparameters

In this section we explore the performance of the hyperparameters related to the embedder. Since our ultimate goal is to create a functioning model which is able to classify newly registered malicious domains, we assess the performance of the hyperparameters by the average precision of the classifier which was trained on the embeddings. The Doc2Vec hyperparameters we examine are the following:

- Embedding Dimension: Number of dimensions of the embedded vector
- Sentence Length: Number of queries to use during training
- Using Metadata: Whether we want to include other query metadata (such as time, IP version, ...) for the embedder besides the resolver IP address.
- Number of Epochs: Amount of times we go over the training data. After Epoch 0 we have gone over the training data once.
- Distributed Memory (dm): Whether to use Doc2Vec's distributed memory for training, or DBOW. When dm is true, the document vectors are trained in the same vector space as the word vectors. If it is set to false, document vectors and word vectors are in different vector spaces.
- First sentence only: Whether or not to only train embedder with the first sentence (set of queries)

These 6 hyperparameters can have many different values, we choose a wide range of them in order to ensure we find the optimal ones:

- Embedding Dimension: 10, 25, 50, 100, 200, 250, 300, 500, 1.000
- Sentence Length: 5, 10, 25, 50, 100, 250, 500, 1000, 2000, 5000, 10000, 20000, 50000
- Using Metadata: None, [Country, ASN], All ([ipv, time, num labels, qtype, country, asn])
- Number of Epochs: 0, 1, 2, 3, 4, 5, 6
- Distributed Memory: True, False
- First sentence only: True, False

In order to properly test the impact of the hyperparameters on the performance, we use default hyperparameters which are used across all tests. This means that if we test for example the use of distributed memory, we generate 2 embedders with the same default settings, apart from dm, which is once true and once false. We have selected the following values as the default hyperparameters.

- Embedding Dimension: 50
- Sentence Length: 1000
- Using Metadata: None
- Number of Epochs: 3
- Distributed Memory: False
- First sentence only: False

Now that we have all possible hyperparameter values as well as the default hyperparameters, we can start training the embedder and classifiers.

5.2.2. Embedding Hyperparameter Results

The performance of the three different classifiers for different hyperparameters is shown in Figure 5.1. It depicts the average precision of the three different classifiers, which are using embedders trained on the 6 different embedding hyperparameters.

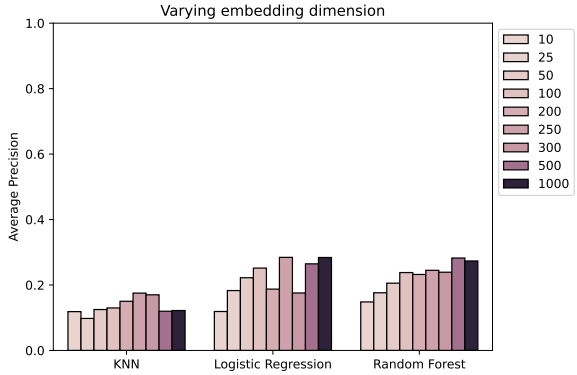
Dimension

When analyzing Figure 5.1a, we can see that the best performance for the dimension is 250 using logistic regression. The surrounding dimensions of 200 and 300 do not seem to perform as well. While 100 does perform almost as well as 250 for the logistic regression and KNN, the improve performance for 250 is enough to warrant using that instead of 100.

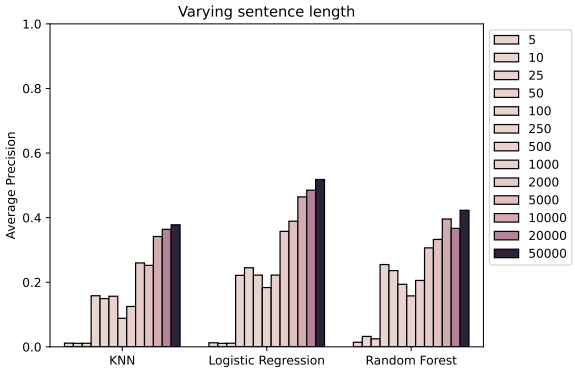
Sentence Length

For Figure 5.1b, the best performance is at the largest sentence length, which does make sense. The more information the classifier has, the better its prediction will be. The drawback of choosing a really high sentence length however, is that it will take much longer to reach this point. Our goal is to detect maliciously registered domains before Netcraft, however if we wait until we reach 50.000 queries, this is very unlikely to happen. There is a trade-off between the number of queries we use to make a prediction, and the amount of time it takes for that prediction.

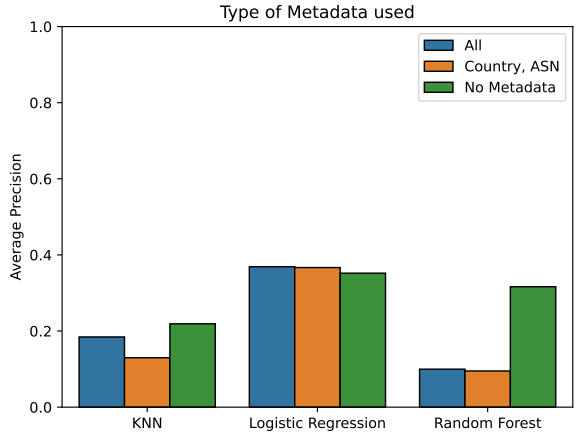
In order to better gauge which sentence length is better, we analyze how many maliciously registered domains will have already been detected after a certain amount of queries. We show this in Figure 5.2, where the cumulative amount of queries is depicted per malicious



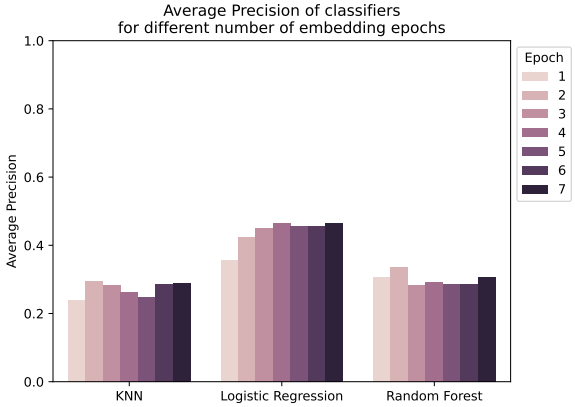
(a) Dim



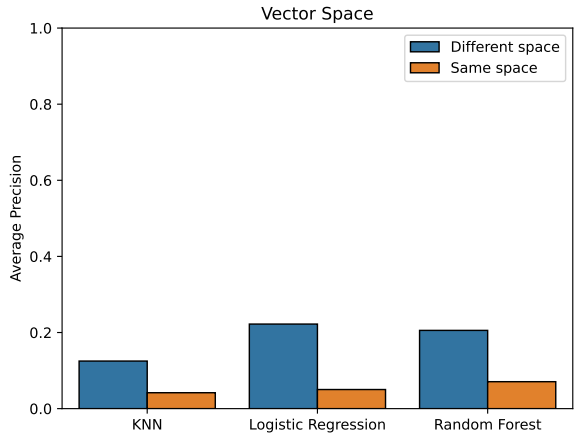
(b) Sentence Length



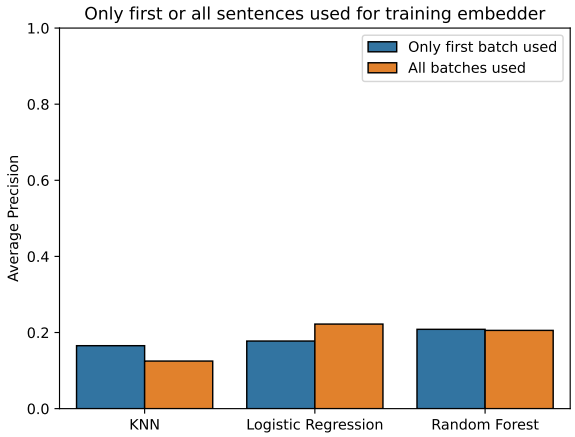
(c) Use Metadata



(d) Number of Epochs



(e) Use Distributed Memory



(f) First sentence only

Figure 5.1: Classifier performance for different embedding hyperparameters

domain. This shows us the percentage of malicious domains which have been reported after each step in the number of queries.

For example if we choose to predict the maliciousness of a domain after 100 queries, if we extend this line up to our plotted cumulative curve, we get to a y value of 0.1193. This means that 11.93% of maliciously registered domains were reported before reaching 100 queries. We want to balance the performance of the classifier (average precision) with the percentage of malicious domainnames which can be

We chose to depict the most ideal sentence lengths for the logistic regression classifier in terms of average precision in Figure 5.1b, namely 100, 2.000 and 10.000. We can see that the average precision for the classifier using 100 queries is 0.25, with 11.93% of domainnames being reported before. While this means that the classifier would still be in time to predict 88.07% of of the maliciously registered domains, we would ideally want a higher average precision. When we increase the amount of queries to 2,000, we also increase the average precision to 0.36, with 27.98% of domains being reported before the classifier is able to do so. This still leaves over two thirds of malicious domain names, while significantly increasing the average precision. If we further increase the number of queries necessary to make a prediction to 10,000, the AP again increase, this time to 0.46. However, this means the classifier is too late for over half the malicious domains.

In this case the middle value of 2,000 queries seems to be the most optimal, since it has a relatively high average precision, while still being soon enough to evaluate over two thirds of the malicious domains.

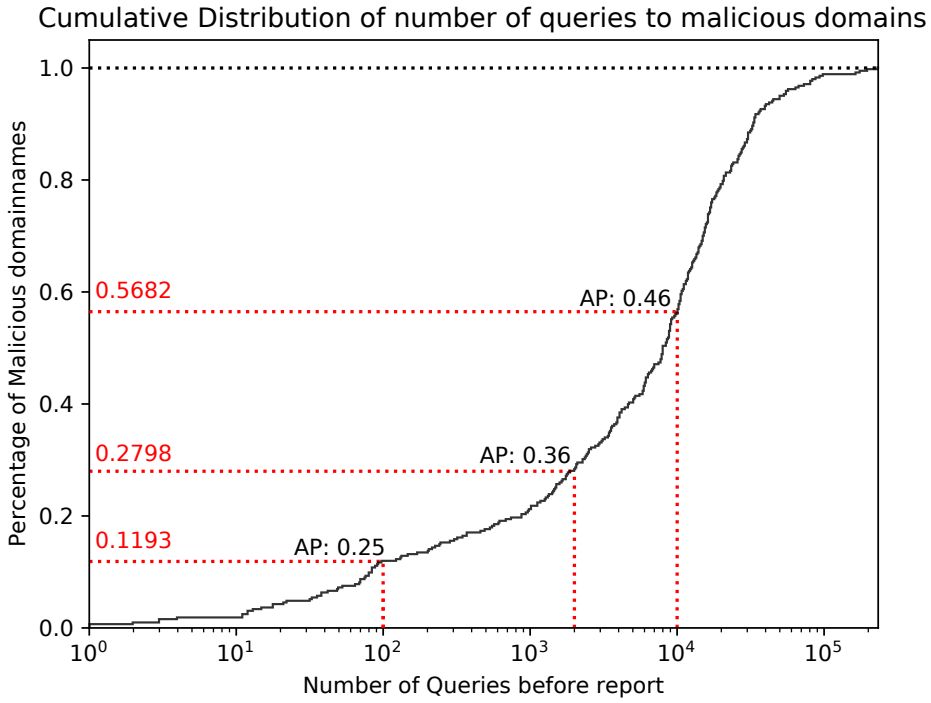


Figure 5.2: Cumulative Distribution of number of queries to malicious domains. The average precision of the logistic regression classifier is marked for 100, 2.000 and 10.000 queries

Metadata

Since we are also interested in the performance of metadata on the embeddings of the domains, we compare no metadata with the country and ASN, as well as all metadata that we selected in Section 4.3 (IPV, Time, Number of Labels, Qtype, Country, ASN). The performance of the different metadata configurations is shown in Figure 5.1c and suggests that there is no real benefit to adding metadata for logistic regression and KNN, while adding metadata seems to actively worsen the result of the Random Forest. For this reason we choose to not include any other information about the queries for the embedder besides just the set of resolver IP addresses.

Number of Epochs

Another hyperparameter we analyze is the optimal number of epochs for the embedder, which we have tested from 1 to 7 in Figure 5.1d. The best performing KNN and Random Forest are for some reason after 2 Epochs. The overall best performing classifier is logistic regression with any epoch of 4 or over. We choose an epoch of 5 to ensure we truly go over all data often enough, especially when we later increase the amount of training data by using all available domainnames and training over a longer date range.

Distributed Memory

We also explore the performance of the classifiers when the embedder is trained with and without distributed memory, see Figure 5.1e. When distributed memory is set to true, the document (domain name) embeddings are put into the same vectors space as the word embeddings (resolvers). The graph shows a very clear difference in performance across all three classifiers when training the embedder with and without dm. The classifiers perform significantly better when dm is not used.

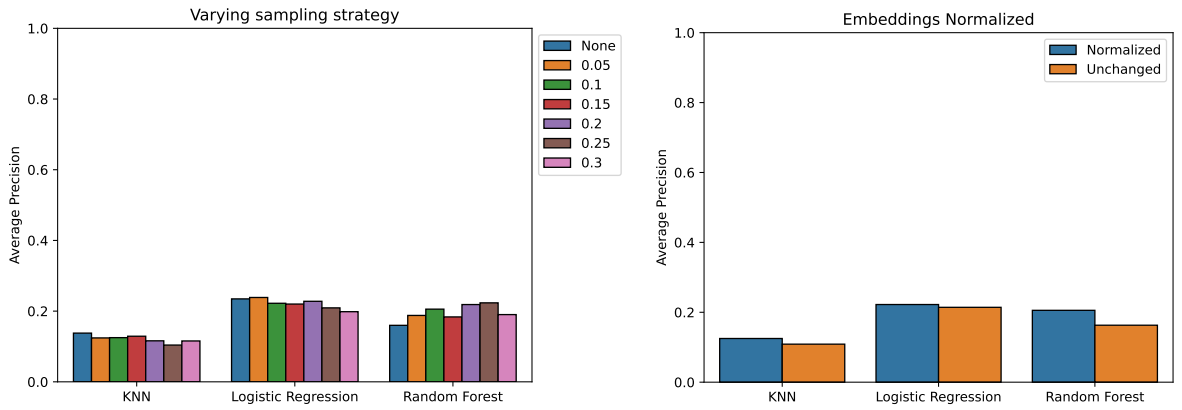
First Sentence Only

We now look at the final embedding hyperparameter that we have explored, whether or not to only use the first sentence of a domain to train the embeddings. In other words, do we only use the first 1.000 queries to each domain to each embedder, or do we use all of the available queries? When analyzing the results of this experiment in Figure 5.1f, we can see that this does not seem to have an impact on the Random forest, but it does change for the KNN and Logistic Regression. While only using the first sentence does seem to be better for the KNN, using all queries works better for the Logistic Regression, which also performs better than the KNN. For this reason we choose to train the embedder using all available queries.

5.2.3. Classifier Hyperparameters

Besides embedding hyperparameters there are also hyperparameters for the classifiers:

- Sampling Strategy: Undersampling of datapoints belonging to the majority class in order to decrease the existing class imbalance. The sampling strategy dictates how many samples of the majority class should be sampled compared to the minority class. For example, if there are 10 malicious domains and we want to sample 250 benign domains, then the sampling strategy would be $10/250 = 0.04$. Only the
- Normalized: This hyperparameter decides whether we normalize the embedding vectors before training the classifier or not



(a) Sampling Strategy

(b) Normalized

Figure 5.3: Classifier performance for different classifier hyperparameters

Sampling Strategy

Figure 5.3a shows that the sampling strategy largely does not have an impact on the average precision of the classifier. It shows that reducing the amount of benign domains in the training dataset does not have a significant impact on the classifier's ability to detect the maliciously registered domains. For this reason we do not apply any undersampling in our final classifier.

Normalizing

The second and final classifier hyperparameter that we analyze is whether or not to normalize the embeddings. When looking at Figure 5.3b, we can tell that the difference in average precision when normalizing and not normalizing is not very big. However in the case of all three classifiers, normalizing the vectors does seem to slightly improve performance, which is why we choose to ultimately normalize the vectors.

5.2.4. Chosen Hyperparameters

Now that we have explored the performance of various different hyperparameters, we have the most optimal value for each one, which is shown in Table 5.1

Hyperparameter Type	Hyperparameter	Value
Embedder	Dimension	250
	Sentence Length	2,000
	Use Metadata	False
	Number of Epochs	5
	Distributed Memory	False
	First sentence only	False
Classifier	Sampling Strategy	None
	Normalized	True

Table 5.1: Chosen Hyperparameter values for final classifier.

6

Final Classification Method

Now that we have explored the optimal hyperparameters for our embedder and classifier, we can start building our final embedder and classifier. We first need to identify which of the 3 types of classifiers we have evaluated to use.

When looking at Figure 5.1 and Figure 5.3, it is clear that in every case Logistic Regression outperforms KNN and Random Forest. To verify this we plot the mean average precision of the three different classifier across all experiments, which is shown in Figure 6.1.

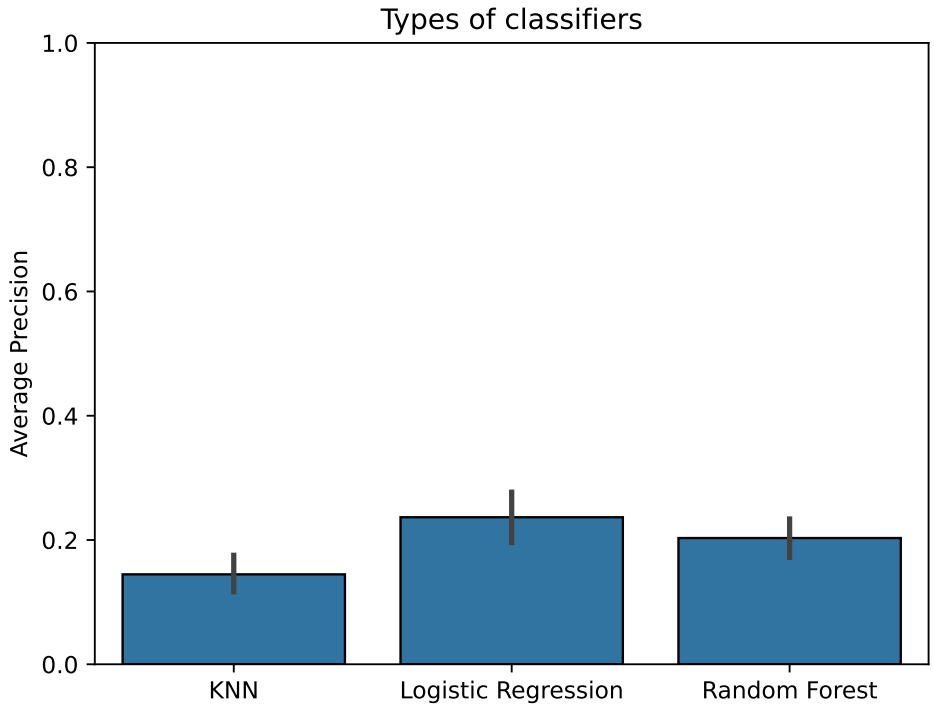


Figure 6.1: Average Precision of KNN, Logistic Regression and Random Forest across all experiments.

The graph clearly confirms that Logistic Regression outperforms the other two classifiers. Furthermore, when we apply all the optimal settings for the three different classifiers, we get

an AP of 0.3520 for Logistic Regression, 0.3164 for Random Forest and 0.2190 for KNN. This confirms that Logistic Regression is in fact the best classifier in our use-case.

6.1. Data

Now that we know the best classifier type as well as the best hyperparameters, we can apply the embedder and classifier on data over a much longer time-frame. For this we use a new registration dataset, spanning from the first of August 2023 until the 23 September 2024.

In order to create the final embedder and classifier, we model the training and test data as close to the real-world deployment scenario as possible. This means that we include all benign and malicious domains during the mentioned timeframe of 13 months. This leaves us with a total of 958,282 benign domains, 1,352 compromised domains (reported after 30 days of registration, counted as benign) and 2,118 maliciously registered domains (0.22%).

With this list of domain names registered within our given timeframe, we can collect all queries sent within 10 days of registration for each of these domains. Any queries that were sent after the domain was detected by netcraft is discarded as to not give the embedder and classifier invalid information.

6.2. Training time-frame

In order to find out the most optimal amount of months to train the classifier on, we train different classifiers on the same embedder for timeframes ranging from 1 to 11 months. The embedder is trained on 11 months of query data (from 2023/08/01 to 2024/07/01). To more closely emulate results we would obtain in a real-world scenario, we test the embedder and classifier on test data from months after the training data. In our case we use 2 months of registration data purely for testing (from 2024/07/01 to 2024/09/01), which ensures that the classifier actually predicts data it has never seen before.

All classifiers use the same embedder to generate the domain embeddings, the only thing that changes is at what point the training data for the classifier starts. The classifier with one month of training data is trained on embeddings of domains registered in June 2024. The classifier with two months of training data is trained on embeddings of domains registered in May or June 2024, and the classifier with 11 months of training data from August 2023 until June 2024 (included). These 11 classifiers are used to make predictions on the same 2 new months of data (from 2024/07/01 to 2024/09/01), the results of which are plotted in Figure 6.2.

This shows that the average precision for the classifiers trained on 1 to 3 months of data perform significantly lower than classifiers trained on 4 months onward. For some reason the classifier which was trained on 4 months of data stands out, since there is a very large jump from 3 to 4 months, with 3 having almost no average precision at all, and 4 months being the highest of all classifiers tested.

One possible explanation for this could be that at that month (March 2024) there were simply more maliciously registered domains that in the months after. To verify this, we can examine the number of benign and malicious registrations over the course of our registration data, shown in Figure 6.3. It shows that there is indeed a significant increase in malicious domain name registrations around March 2024. This is also apparent when looking at the ratio of

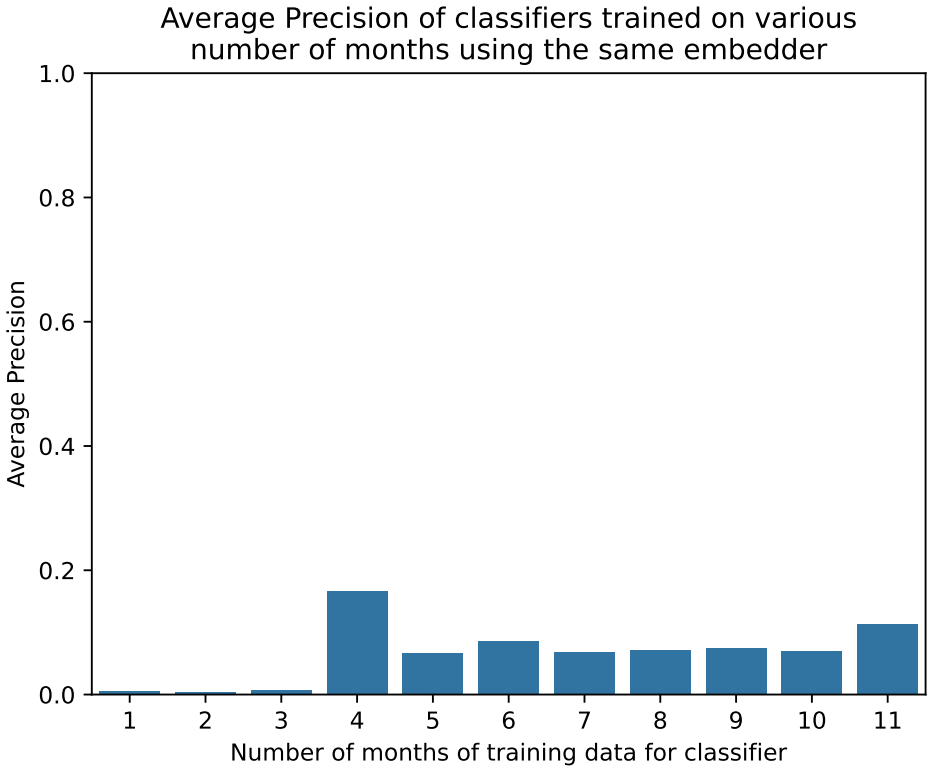


Figure 6.2: Average Precision of classifiers with different training time-frames using the same embedder.

malicious domains in the different training sets (see Figure 6.4).

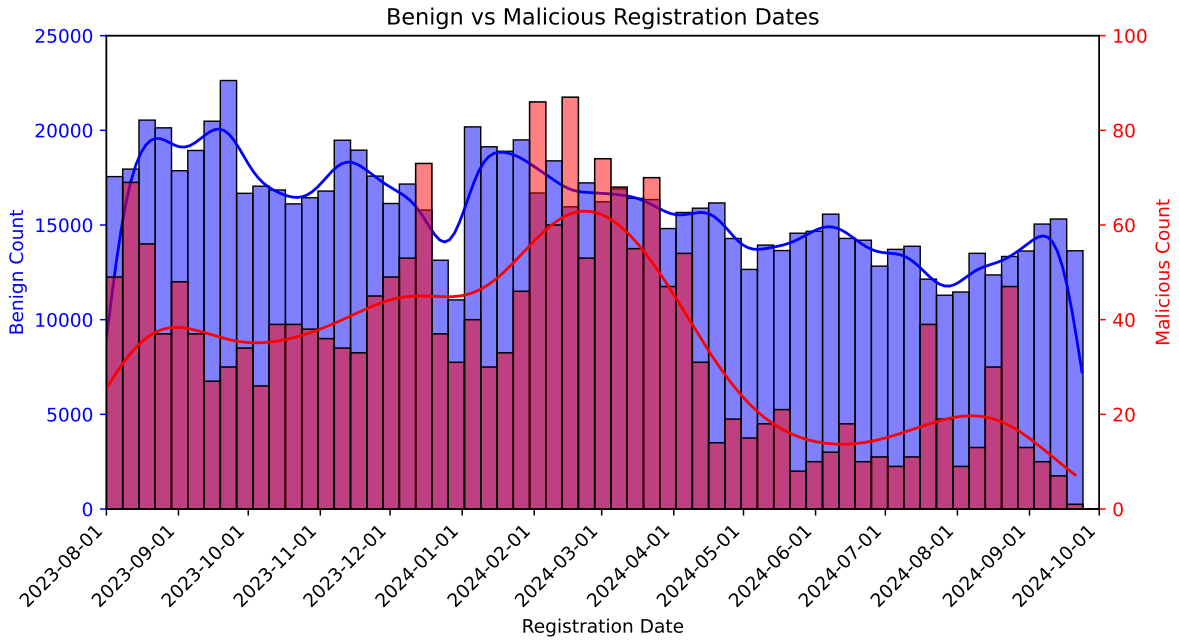


Figure 6.3: Number of benign and malicious registrations from August 2023 until September 2024.

One would expect the performance of the classifier to further increase as the ratio of malicious domains increases at 5 months, however this is not the case. This could be because the

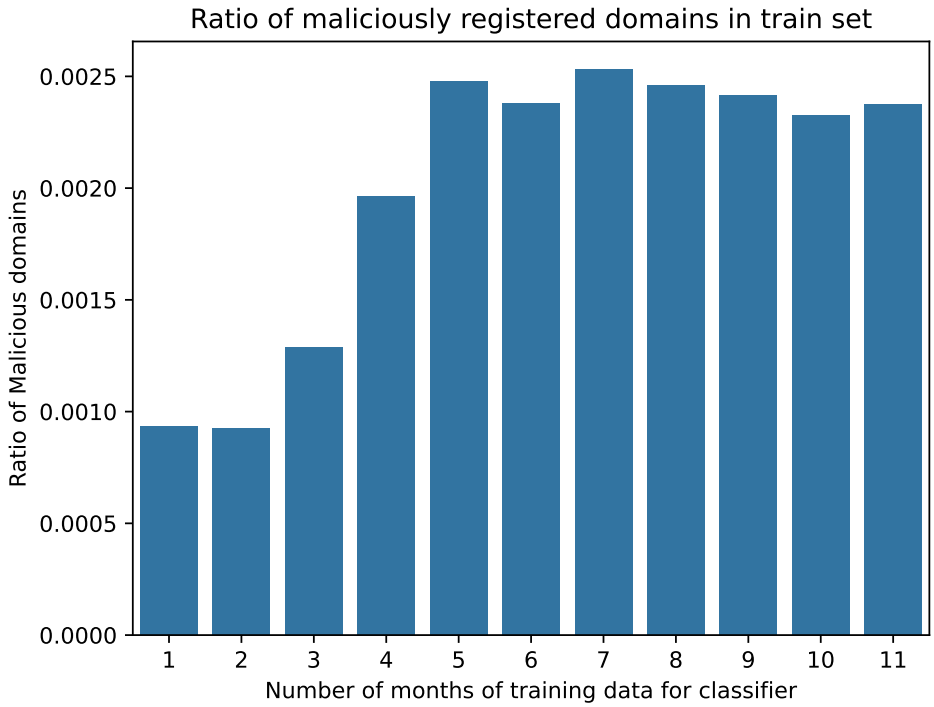


Figure 6.4: Ratio of maliciously registered domains in the training set.

extra month of data is not as useful because it is so far away from the testing months.

An issue this does reveal, is the fact that if the amount of maliciously registered domains decreases, the embedder and classifier will have a harder time detecting new malicious domains. Regardless of the reason for this dip in malicious registrations, we can conclude that 4 months is in our case the most optimal time-frame for training the classifier.

Now that we know that 4 months is the optimal amount for the classifier, we can explore whether we can also train the embedder on 4 months of training data. To do this we use the embedder from the previous experiments (trained from August 2023 until June 2024 included), and train a new embedder trained on only 4 months of query data (from May 2023 until June 2024 included). We train a classifier using the domain embeddings of both embedders, and test their performance on data from July 2024. The average precision of both of these classifiers is 0.17, suggesting that increasing the time-frame of the embedder far into the past does not improve the classifier's predictions.

For this reason we choose our final method to use Doc2Vec trained on 4 months of query data, with the Logistic Regression also being used on those same 4 months of data.

6.3. Performance

Our final classifier has an average precision of 0.17 for the training months of 2024/03/01-2024/07/01 and the test months of 2024/07/01/2024/08/01. We plot the precision-recall graph of our classifier in Figure 6.5, which shows the precision and recall of the classifier as the threshold is changed. The closer the curve is to the bottom left corner, the worse the performance; the closer it is to the top right corner, the better.

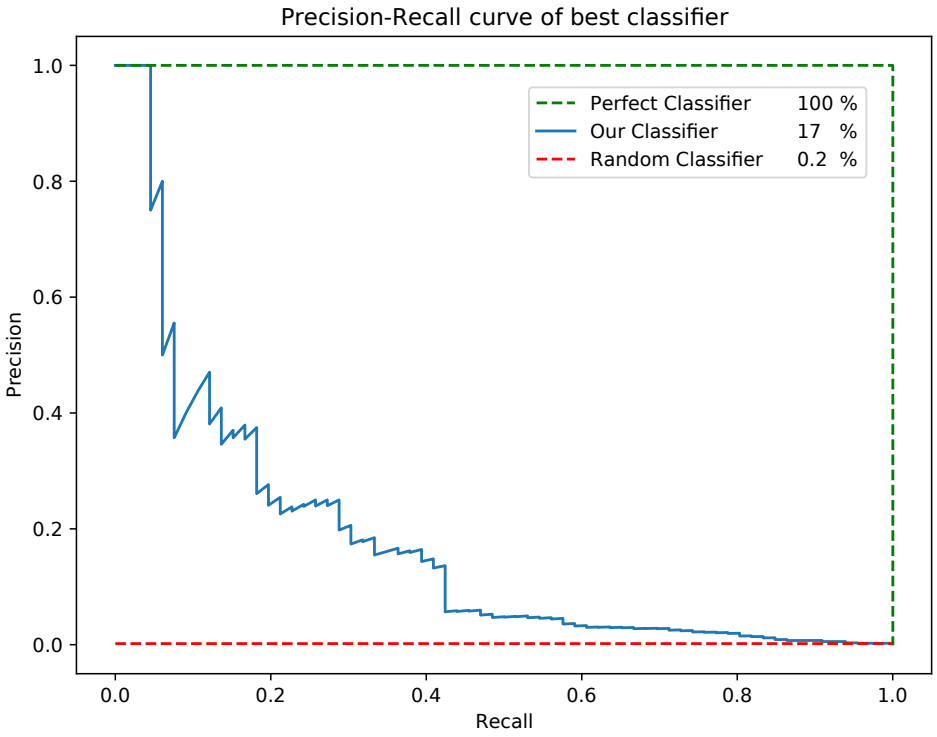


Figure 6.5: Precision-Recall graph of our best classifier, a theoretical perfect classifier and a random classifier.

A perfect classifier would have a consistent precision of 1 as the threshold decreases, since the classifier would be able to perfectly separate the malicious domains from the benign domains. Then once the threshold is at the point where it has captured all malicious domains (Recall of 1), decreasing the threshold further will only cause more benign domains to be flagged as well.

On the flip side, a random classifier's precision will always be around the malicious domain ratio, which is 0.2%. As the threshold is lowered, the precision remains the same, but the recall increases since more malicious domains are being flagged.

While the precision recall curve of our classifier is much closer to the bottom left corner than the top right corner, this does not mean that it is a bad classifier. A perfect classifier is practically unattainable in any real-world scenario, and our classifier performs significantly better than a random classifier would. This proves that detecting maliciously registered domains based on just the DNS query traffic is possible. Our method is not a silver bullet, as it still flags many benign domains, and misses many malicious domains, however it could be a useful tool in helping DNS registries combat malicious registrations.

Another thing to keep in mind is that we are assuming that the Netcraft data we are using is correct, however in practice this is not the case. There are bound to be False Negatives in Netcraft, and potentially even False Positives. Unfortunately we are not in a position to explore to what extent this is the case for this thesis, however this could be valid exploration for future work.

6.4. Deployment Scenario

As mentioned previously, the classifier is not able to perfectly detect all malicious domains, however it could be used as another tool to help prevent malicious activities online.

One way this could be achieved is by simply sending the classifier's prediction to registry staff. So instead of sending a verdict of whether the domain is maliciously registered or not, we cna instead just send the likelyhood of a domain being malicious. This can be combined with other metrics to help registry staff make more informed decisions on which domains to investigate.

This approach seems to be more sensible than simply giving a malicious/benign verdict, as it offers more nuance and flexibility to staff and lowers the risk of false flags.

However for the sake of gaining a better understanding of the classifier's performance, we find an optimal threshold and then explore how the classifier would perform.

6.4.1. Selecting Threshold

In order to see how the classifier would perform, we first need to select a threshold which has a good balance of precision and recall. We do not want our precision to be too low, since investigating malicious domains takes resources, and we generally want to avoid false positives. We also want to have a high recall so we do not miss too many malicious domains, however we do put slightly more importance on the precision than the recall. A good precision in this case is around 1/3, which would put the recall around 0.2. We show the exact spot in Figure 6.6.

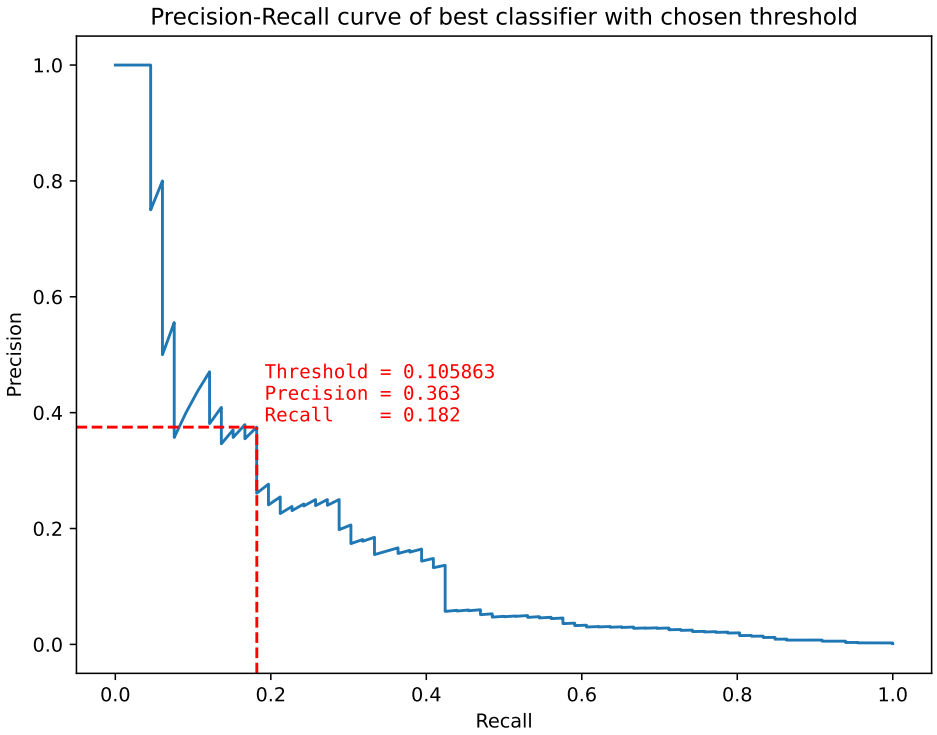


Figure 6.6: Precision-Recall graph of our best classifier with selected threshold.

Our chosen threshold is around 0.1059, which gives us a Precision of 0.375 and a Recall of

0.182. We have applied this classifier and threshold to the query data from July 2024, which had 49,358 registrations, of which 66 were malicious. The classifier predicted 33 domains as malicious, of which 12 were actually malicious. This means that the classifier missed 54 malicious domains.

6.4.2. Detected Malicious domains timeline

Now that we have the threshold we can also explore the time-frame of the predictions, as in how much sooner is the classifier able to detect a malicious domain compared to Netcraft. We show at which point the classifier is able to detect the malicious domains (at what point 2,000 queries are reached), and compare it to the time it took Netcraft to report (see Figure 6.7).

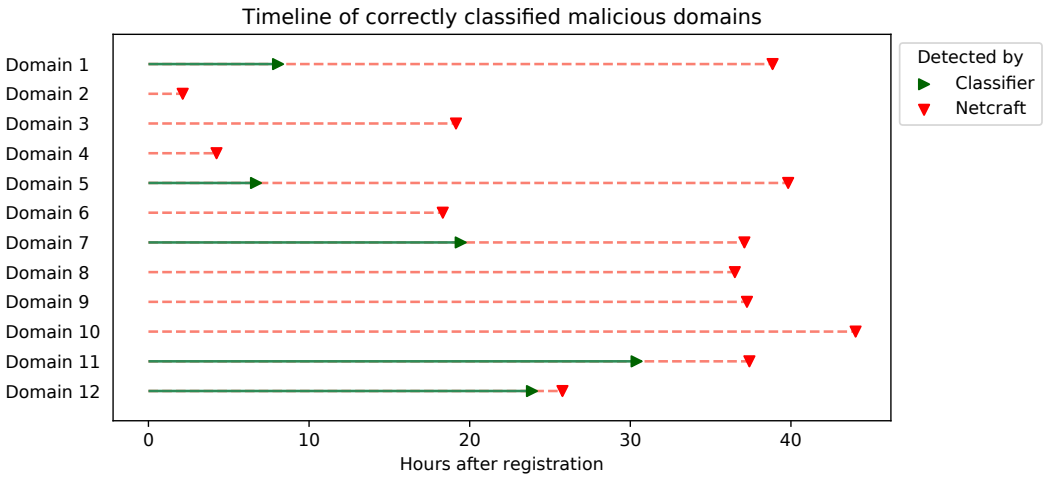


Figure 6.7: Timeline of classifier for correctly classifier malicious domains.

The graph displays the duration it takes for the classifier to detect the domain in green, and the time it took Netcraft to report them. In the cases where there is no green line, this means that Netcraft was able to detect the malicious domain before it has received 2,000 queries. This means that for 7 out of the 12 domains, the classifier was still able to correctly classify the domain as malicious, but not sooner than Netcraft because the 2,000 queries were not reached.

For 5 of the domains however the classifier was able to detect the domain as malicious, in most cases much sooner than Netcraft. Netcraft detected these domains on average after 35 hours, while the classifier was able to detect these in just 17 hours. This means that the classifier essentially halved the amount of time the malicious domain is able to operate, a difference of 18 hours. In the case of Domain 5, it is able to reduce the detection time by 33 hours. Considering each second is important when it comes to shutting down these malicious domains, this is a positive result.

One way this could be improved even further is by training multiple classifiers, which predict the maliciousness of the domains at different points in time. This way we could have one classifier after say 50 queries, one after 250, one after 2,000, and so forth. This would greatly increase the amount of malicious domains that could be detected. Implementing these subsequent classifiers may be possible to do on the same classifier, which would significantly reduce the training overhead. This would be a very interesting step for future work.

7

Conclusion

In this thesis, we proposed using word embeddings on DNS data in order to detect maliciously registered domain names. In order to achieve this, we first explored how DNS Query data can be represented in a latent space in Chapter 4. We explored different possibilities for embedding domain names and whether Metadata could improve the embeddings.

The next chapter (Chapter 5) explored the optimal classifier and hyperparameters, answering the second research sub-question.

In Chapter 6 we created the final classifier to answer the last sub-question and assessed its performance in a scenario more closely mimicking deployment. We examined the precision recall graph of the classifier, which proved that the classifier is able to use the embeddings of domain names in order to make predictions on the maliciousness of the domains. It achieves an average precision of 17% in our realistic detection scenario, which amounts to a precision of 36% and a recall of 18% for our chosen threshold, only using the resolver IP addresses as input for the embedder.

We also investigated the timeline of the correctly identified malicious domains, which showed that the classifier is able to detect malicious domains significantly faster than Netcraft for half of the correctly detected malicious domains.

All of the above allows us to answer our main research question: "How can we apply a combination of representation learning and classification algorithms to detect newly registered malicious domains?". This is possible by using document embeddings to embed domain names using their DNS traffic, which are then used by a logistic regression classifier to predict whether the domain is maliciously registered or not.

In conclusion, while this approach is not fitted to make predictions on the maliciousness of domains by itself, and it cannot detect all malicious domains, it can be a useful tool to build upon existing detection methods, or to aid humans in the detection process. SIDN Labs is working on deploying the method presented in this thesis at SIDN, in order to detect maliciously registered domains in the ".nl" zone sooner.

7.1. Future Work

There are many different ways in which this work can be extended and continued. In this section we cover the avenues we have identified, but ended up not pursuing due to time and resource constraints.

7.1.1. Additional Features for the classifier

Our final classifier strictly uses the embeddings generated for each domain as input. However as other papers have shown [2, 18, 24], some features from DNS query data can also be very indicative of the maliciousness of domains. Adding additional features from these papers to the classifier could lead to better results, as it would receive more context and information for each domain.

7.1.2. Subsequent Classification

One way our method could be further improved is by sequentially predicting the maliciousness of domains. For example instead of only making a prediction after 2,000 queries like in our final classifier, we can instead make a prediction at multiple spots, for example after 200, 500, 2,000, 10,000 queries. This allows for more malicious domains to potentially be detected, without sacrificing precision.

It might even be possible to use a single embedder to embed all of domains at each different stage, however that would have to be examined.

7.1.3. Detecting Compromised Domains

One potential avenue for future work is applying embeddings and a classifier not on maliciously registered domains as done in this thesis, but instead apply it on existing domains in order to detect compromised domains.

This relies on the assumption that compromised domains have a change in traffic that is reflected in the embeddings of the domains.

If this is the case, then one way this could be achieved is by training an embedding model every so often (for example every day) and comparing the change in embeddings for the same domain. If the embeddings differ significantly, this might be an indication that a compromise of the domain took place. The reason we did not examine this further is because this is much more resource intensive. It would require continuous training using all query data to all domains, which amounts to about 2-3 billion queries a day in the ".nl" zone.

7.1.4. Deeper Results Analysis

Another avenue for future work is examining the False Positives and False Negatives more closely. This would provide more insight into why the classifier and embedder may not be performing optimally, and how this could be improved. This analysis would be challenging, as the classifier only receives a vector as input, which makes it nearly impossible for humans to figure out if there is an issue with this embedding.

Furthermore, the Netcraft dataset is not perfect. There are bound to be some domains which are mislabeled, such as malicious domains being considered benign and vice-versa. It might be worth exploring whether any of the false positives reported by our classifier are actually

malicious domains that went undetected by Netcraft, or whether any false negatives are actually benign domains.

8

Acknowledgements

There are many different parties that have enabled me to write this thesis. From my supervisors both at TU Delft and at SIDN, as well as SIDN itself, my family and my friends, all have played an important role in helping me write this thesis.

I would like to especially thank Giovane Moura, Thymen Wabeke, Thijs van der Hout and Georgios Smaragdakis for supervising my project and helping me along at every step. Giovane has opened the door for me, putting me in touch with SIDN helping me select a suitable thesis project as well as giving me good advice throughout my writing process of the thesis. Georgios has been very supportive throughout my thesis and has helped me tremendously in advancing my thesis with his extensive knowledge and experience.

I would like to thank SIDN for allowing me the chance to work on this project, allowing me to gather a lot of valuable work experience and for being patient and granting me extensions to ensure I can properly finish my project.

I would also like to thank everyone at SIDNLabs for being so welcoming, kind, supportive and always open to share their expertise. I would like to especially thank Thymen and Thijs for supervising and guiding me throughout the project, helping me at every step whenever I required assistance, even when they themselves have a lot of other work and responsibilities.

I would also like to thank my close friends, Pit Feltes, Prakhar Jain, Dylan Bravo, Niklas Perujo, Alexander Schnapp, Frank Broy, Kristof Sandor, who have all been very supportive throughout this process and helped me continue

I would also like to thank my family, especially my parents and my sister, for the continuous moral and financial support throughout not only the thesis, but my entire life. They were always willing to help when I needed it, be that in High school, when applying and moving to Delft as well as when I moved again during my thesis internship.

All of the people I mentioned here, and many more, have played an important role in me getting where I am now. I have gained a lot of experience and would like to think I have grown as a person because of them, and I would not have been able to finish my thesis without them. I am very grateful to have had their support.

References

[1] Alan Akbik, Duncan Blythe, and Roland Vollgraf. "Contextual String Embeddings for Sequence Labeling". In: *COLING 2018, 27th International Conference on Computational Linguistics*. 2018, pp. 1638–1649.

[2] Manos Antonakakis et al. "Building a dynamic reputation system for {DNS}". In: *19th USENIX Security Symposium (USENIX Security 10)*. 2010.

[3] Manos Antonakakis et al. "Detecting malware domains at the upper {DNS} hierarchy". In: *20th USENIX Security Symposium (USENIX Security 11)*. 2011.

[4] Leyla Bilge et al. "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis." In: *Ndss*. 2011, pp. 1–17.

[5] Piotr Bojanowski et al. "Enriching Word Vectors with Subword Information". In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146. ISSN: 2307-387X.

[6] Daiki Chiba et al. "DomainProfiler: Discovering domain names abused in future". In: *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE. 2016, pp. 491–502.

[7] Hyunsang Choi, Heejo Lee, and Hyogon Kim. "BotGAD: detecting botnets by capturing group activities in network traffic". In: *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*. 2009, pp. 1–8.

[8] T. Cover and P. Hart. "Nearest neighbor pattern classification". In: *IEEE Transactions on Information Theory* 13.1 (1967), pp. 21–27. DOI: 10.1109/TIT.1967.1053964.

[9] David R Cox. "The regression analysis of binary sequences". In: *Journal of the Royal Statistical Society Series B: Statistical Methodology* 20.2 (1958), pp. 215–232.

[10] European Union Agency for Cybersecurity. *ENISA Threat Landscape 2023*. https://www.enisa.europa.eu/publications/enisa-threat-landscape-2023. 2023.

[11] facebookreasearch. *StarSpace Repository*. Dec. 2019. URL: https://github.com/facebookresearch/StarSpace.

[12] facebookresearch. *fastText Repository*. Mar. 2024. URL: https://github.com/facebookresearch/fastText.

[13] flairNLP. *Flair Repository*. Oct. 2024. URL: https://github.com/flairNLP/flair.

[14] Mihajlo Grbovic and Haibin Cheng. "Real-time personalization using embeddings for search ranking at airbnb". In: *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*. 2018, pp. 311–320.

[15] Mihajlo Grbovic et al. "E-commerce in your inbox: Product recommendations at scale". In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2015, pp. 1809–1818.

[16] Casper Hansen et al. “Contextual and sequential user embeddings for large-scale music recommendation”. In: *Proceedings of the 14th ACM Conference on Recommender Systems*. 2020, pp. 53–62.

[17] Casper Hansen et al. *Contextual and Sequential User Embeddings for Music Recommendation*. Apr. 2021. URL: https://research.atspotify.com/2021/04/contextual-and-sequential-user-embeddings-for-music-recommendation/.

[18] Shuang Hao et al. “PREDATOR: proactive recognition and elimination of domain abuse at time-of-registration”. In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016, pp. 1568–1579.

[19] Wenxuan He et al. “Malicious domain detection via domain relationship and graph models”. In: *2019 IEEE 38th International Performance Computing and Communications Conference (IPCCC)*. IEEE. 2019, pp. 1–8.

[20] Tin Kam Ho. “Random decision forests”. In: *Proceedings of 3rd international conference on document analysis and recognition*. Vol. 1. IEEE. 1995, pp. 278–282.

[21] Lambda-3. *Indra Repository*. Dec. 2018. URL: https://github.com/Lambda-3/Indra.

[22] Quoc Le and Tomas Mikolov. “Distributed representations of sentences and documents”. In: *International conference on machine learning*. PMLR. 2014, pp. 1188–1196.

[23] Jehyun Lee and Heejo Lee. “GMAD: Graph-based Malware Activity Detection by DNS traffic analysis”. In: *Computer Communications* 49 (2014), pp. 33–47.

[24] Samaneh Mahdavifar et al. “Classifying malicious domains using DNS traffic analysis”. In: *2021 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*. IEEE. 2021, pp. 60–67.

[25] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[26] Paul Mockapetris. *Domain names - concepts and facilities*. RFC 1034. IETF, Nov. 1987. URL: http://tools.ietf.org/rfc/rfc1034.txt.

[27] Giovane C. M. Moura et al. “Characterizing and Mitigating Phishing Attacks at ccTLD Scale”. In: *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24)*. Salt Lake City, UT, USA: ACM, Oct. 2024, p. 15. ISBN: 979-8-4007-0636-3/24/10. DOI: 10.1145/3658644.3690192.

[28] Netcraft. *Malware Feeds and Cyber Threat Intelligence*. https://www.netcraft.com/cybercrime/malicious-site-feeds/. 2023.

[29] Jeffrey Pennington, Richard Socher, and Christopher D Manning. “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[30] Roberto Perdisci, Igino Corona, and Giorgio Giacinto. “Early detection of malicious flux networks via large-scale passive DNS traffic analysis”. In: *IEEE Transactions on Dependable and Secure Computing* 9.5 (2012), pp. 714–726.

[31] Roberto Perdisci et al. “Detecting malicious flux service networks through passive analysis of recursive DNS traces”. In: *2009 Annual Computer Security Applications Conference*. IEEE. 2009, pp. 311–320.

[32] piskvorky. *Gensim Repository*. Aug. 2024. URL: https://github.com/piskvorky/gensim.

[33] Radim ehek and Petr Sojka. “Software Framework for Topic Modelling with Large Corpora”. English. In: *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. http://is.muni.cz/publication/884893/en. Valletta, Malta: ELRA, May 2010, pp. 45–50.

[34] Magnus Sahlgren. “The distributional hypothesis”. In: *Italian Journal of linguistics* 20 (2008), pp. 33–53.

[35] Juliano Efson Sales et al. “Indra: A Word Embedding and Semantic Relatedness Server”. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. Miyazaki, Japan: European Language Resources Association (ELRA), May 2018.

[36] SIDN. *About SIDN*. Accessed 2024/01/04.

[37] SIDNLabs. *ENTRADA stored data*. Accessed 2024/03/25.

[38] SIDNLabs. *Number of ".nl" domains*. Accessed 2024/11/12.

[39] SIDNLabs. *About SIDNLabs*. Accessed 2024/01/04.

[40] Centraal Bureau voor de Statistiek. *2.2 million cybercrime victims in 2022*. May 2024. URL: https://www.cbs.nl/en-gb/news/2023/19/2-2-million-cybercrime-victims-in-2022.

[41] S. Tajalizadehkhoob. “The Role of Hosting Providers in Web Security. Understanding and Improving Security Incentives and Performance via Analysis of Large-scale Incident Data”. PhD thesis. Organisation & Governance - TPM, TU Delft, 2018. DOI: 10.4233/uuid:c343a2dd-15d1-4921-9b45-f00ee38177d8.

[42] Moritz Müller Thymen Wabeke Thijs van der Hout. *DNS2Vec: applying representation learning to DNS data*. Accessed 2024/04/11. URL: https://www.sidnlabs.nl/en/news-and-blogs/dns2vec-applying-representation-learning-to-dns-data/.

[43] Roberto Turrin et al. “30Music Listening and Playlists Dataset”. In: *Poster Proceedings of the 9th ACM Conference on Recommender Systems, RecSys 2015, Vienna, Austria, September 16, 2015*. Ed. by Pablo Castells. Vol. 1441. CEUR Workshop Proceedings. CEUR-WS.org, 2015. URL: https://ceur-ws.org/Vol-1441/recsys2015%5C_poster13.pdf.

[44] US Federal Bureau of Investigation, Internet Crime Complaint Center. *Internet Crimer Report*. https://www.ic3.gov/Media/PDF/AnnualReport/2023_IC3Report.pdf. 2023.

[45] Flavian Vasile, Elena Smirnova, and Alexis Conneau. “Meta-prod2vec: Product embeddings using side-information for recommendation”. In: *Proceedings of the 10th ACM conference on recommender systems*. 2016, pp. 225–232.

[46] L. Wu et al. “StarSpace: Embed All The Things!” In: *arXiv preprint arXiv:1709.03856* (2017).

[47] Hiroto Yamada et al. “Unknown Malicious Domain Detection based on DNS Query Analysis using Word2Vec”. In: *2022 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE. 2022, pp. 1096–1097.

[48] Yury Zhauniarovich et al. “A survey on malicious domains detection through DNS data analysis”. In: *ACM Computing Surveys (CSUR)* 51.4 (2018), pp. 1–36.

[49] Yury Zhauniarovich et al. “A survey on malicious domains detection through DNS data analysis”. In: *ACM Computing Surveys (CSUR)* 51.4 (2018), pp. 1–36.