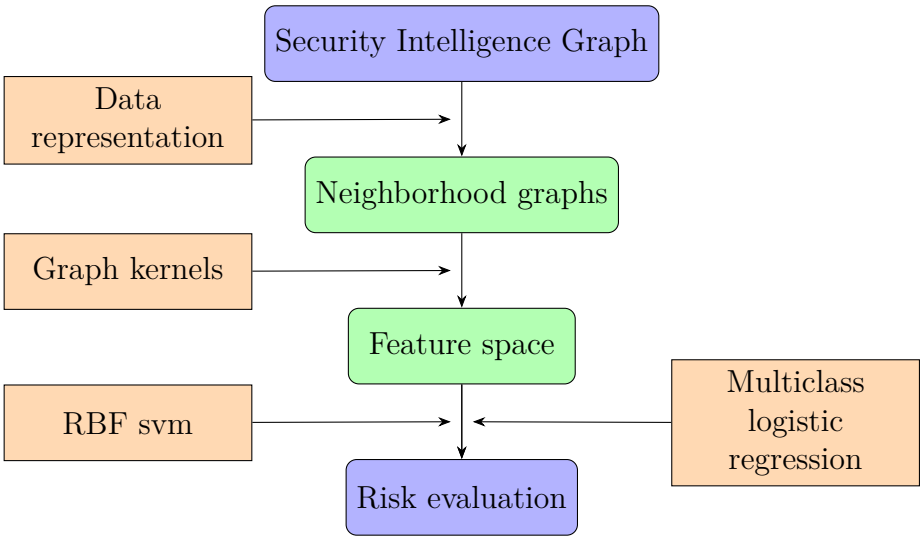
CHALMERS
UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG



Risk Modeling on Cyber Vulnerability Graphs

Modeling of Risk in the Cyber Threat World

Master's thesis in Computer science and engineering

Simon Larsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

Risk Modeling on Cyber Vulnerability Graphs

Modeling of Risk in the Cyber Threat World

Simon Larsson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Risk Modeling on Cyber Vulnerability Graphs
Modeling of Risk in the Cyber Threat World
Simon Larsson

Supervisor: Fredrik Johansson, Department of Computer Science and Engineering
Advisor: Mats Kvarnström, Recorded Future
Examiner: Marina Axelson-Fisk, Mathematical Sciences

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

The figure on the cover page outlines our modeling approach. The Security Intelligence graph is introduced in Section 1.1. The data representation, neighborhood graphs and graphs kernels are described in Chapter 2. Here are also the classification methods radial basis function support vector machine (RBF svm) and multiclass logistic regression outlined. In Chapter 3 a description of how the classification methods were implemented can be found. Moreover the feature space is described there. In Chapter 4 the results of the risk evaluations are shown.

Typeset in LaTeX
Gothenburg, Sweden 2021

Risk Modeling on Cyber Vulnerability Graphs
Modeling of Risk in the Cyber Threat World
Simon Larsson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Cyber security is a field which gets an increasing amount of attention. Record Future is a company providing services to aid and understand this field. An important part of their service is the Security Intelligence Graph (SIG), describing entities in the cyber threat world and how they relate to one and other. This master's thesis proposes a k-neighborhood graph approach for estimating security risks regarding Internet Domains which are present on the SIG. In order to classify Internet Domains their k-neighborhood graphs, on SIG, are mapped to a feature space using Weisfeiler-Lehman graph kernels. Two class and three class classifiers are implemented using support vector machines and logistic regression models. The results suggest that this approach works well for the two class classifier, where the models achieve accuracies around 0.96 and F_1 scores around 0.89. The results are less promising for the multiclass models.

Keywords: cyber security, graph kernels, Weisfeiler Lehman graph kernels, risk classification models.

Acknowledgements

I want to thank my advisor Mats Kvarnström, at Recorded Future, for making this project possible and for providing great guidance and supervision. I also want to thank my supervisor Fredrik Johansson, at the Department of Computer Science and Engineering at Chalmers University, for guiding me through relevant theory and providing invaluable guidance throughout this project.

In addition I want to say thanks to the company Recorded Future for helping me in this project and making it possible, especially to the analytics group at the Gothenburg office. Lastly, I wish to say thanks to friends and family whom have been very supportive during this project.

Simon Larsson, Gothenburg, August 2021

Contents

List of Figures

List of Tables

1

Introduction

Cyber security is a topic which have had growing attention during the last years. The Swedish Government have recently decided that a national center for cyber security will be instituted, among other reasons because of the cost of cyber security breaches is estimated to billions of Swedish crowns (SEK) (hundreds of millions euros) [1]. Moreover Talesh convincingly argues that cyber security risks are considerable both for companies and consumers [2]. This motivates the cyber security problem of classifying Internet Domains as either malicious or benign. Our classification is done by considering the context the Internet Domains are in.

There have been several previous attempts to model cyber security risk [3], [4], [5], [6], [7] and [8]. Tavabi et al [7] classify cyber vulnerabilities as having low or high risk of being exploited. A cyber vulnerability is a vulnerability in software which could be exploited by criminal actors, for example through a ransomware attack. All vulnerabilities are not likely to be exploited especially those without high associated risk. The associated risk for a cyber vulnerability depends on how much damage it could lead to and how much access it gives etc. Another important aspect is if the vulnerability has been found by criminal actors. A way to capture this is by supervising Dark Web forums, an unregulated part of the Internet. Tavabi et al use this approach by learning a feature representation from discussion around cyber vulnerabilities on these Dark Web forums. Lastly they use the feature representation in a support vector machine in order to get a classification model.

The approach of Tavabi et al has several similarities to our approach. Two examples of this are that the context will be used to find an embedding and a support vector machine will be used for classification. There are however some important differences. Firstly, the classifications are done for Internet Domains instead of cyber vulnerabilities. In Carter et al [5], they model risk for Internet Domains but their modeling approach is different compared to ours. They use factor models which we do not use. Secondly, instead of providing a prediction of exploitation our aim is to estimate the risk associated with Internet Domains as it is now rather than in the future. Without the predictive property in our approach, issues with including information which is strongly correlated with an exploitation rather than an indication of a future exploitation do not need to be handled. These issues can be considered as data leakage problems which in our approach are not as important as in a predictive model. Lastly, our approach use graph kernels instead of deep learning approaches making them easier to interpret. An important property in real-world applications.

1.1 Recorded Future and the Security Intelligence Graph

Recorded Future is a cyber security company providing clients with risk evaluations of entities in the cyber threat world along with the context needed for security experts to make their own risk evaluations. The different entity types contained in Recorded Future's data range from companies, nations and organizations to malwares and IP addresses.

Recorded Future provides a risk score for entities between 0 and 99, where scores above 80 are considered critical. The entities which have risk scores are IP addresses, Internet domains, cyber vulnerabilities, companies and hashes (which are related to malwares). Entities which are considered critical have technical evidence which supports the score, hence the risk evaluations of these entities can be considered more accurate than risk evaluations resulting in lower risk scores. This is natural since not having observed something harmful is not the same as something being harmless. One example of technical evidence is when an IP address corresponds to a confirmed command and control server, used for a distributed denial-of-service attack (DDoS-attack). A high risk score for an Internet domain could, for example, be triggered by sightings of malware on the domain. Not all entities have accurate scores. Some entities have low scores although they should have higher so finding these is of interest.

Recorded Futures database is called the Security Intelligence Graph (SIG). This graph aims to capture the whole cyber threat world meaning that all important aspects in a cyber security analysis should be present on the SIG. Hence the SIG is huge and processing it is demanding both through computational and modeling aspects. The graph has two parts, the ontology graph and the event graph, see Figure 1.1. The ontology graph captures more permanent relations such as Berlin being the capital of Germany, but also relations such as companies ownership of Internet domains. The event graph on the other hand aims to capture more fast moving information, such as information about cyber attacks which might only be relevant in a short period of time. The most relevant part of the event graphs are *references*, which contain a list of the mentioned entities present on the SIG along with important security information [9]. It is mainly the event graph that will be considered further.

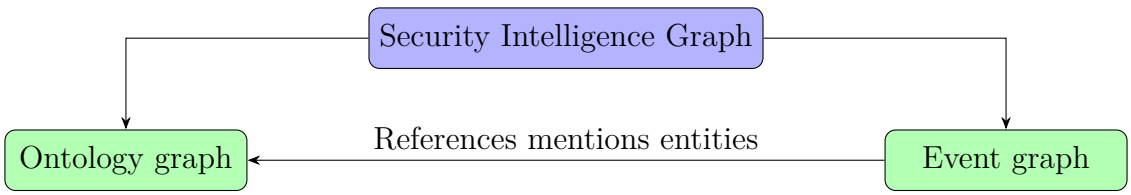


Figure 1.1: Schematic view of the Security Intelligence Graph.

1.2 Problem

Finding potential threats is often contextual and this context is described in the SIG. The context is used for manual annotation and risk evaluation of entities. There are also algorithmic approaches for risk evaluation [9]. However a k-neighborhood graph modeling approach of the context is not implemented. This motivates the following research questions:

1. To what extent can exploiting the network structure on the SIG for Internet Domains improve risk assessments of cyber threats?
2. Can k-neighborhood graphs around Internet Domains be used to classify Internet Domains as either malicious (risk score ≥ 80) or benign (risk score < 80)?
3. What information should be used from the SIG in order to generate k-neighborhood graphs for accurate estimations?
4. Can a more fine-grained estimation be made where the classification is made to more granular risk score intervals?
5. Which graph embedding should be used for the k-neighborhood graphs in order to make classifications?

Since Recorded Future provides risk scores for entities it is possible to answer these questions with their annotated data. If using k-neighborhood graphs can yield accurate risk evaluations similar to the risk rules, it has been shown that these graph interpretations hold valuable information. A possible consequence is that if these approaches are promising, they might lead to additional risk rules in the future. Risk rules are used by Recorded Future to estimate risk scores. More generally, if graph approaches seem promising for modeling cyber security threats regarding Internet Domains using Recorded Future's data, similar modeling approaches could be promising for publicly accessible data as well. These questions will be answered by constructing different classification models.

1.3 Limitations

For technical reasons the entities which are to be classified are Internet domains. This choice is motivated by issues which stems from data collection, namely without DNS information too few of member of the other entity types got a meaningful k-neighborhood graph. Other entity types are included in the data set though, but only for classifying Internet domains correctly.

Only a subset of the records available in the data base was used. One reason for this is the size of the data base, it contains hundred of millions entities and more than 60 billion references [9]. Hence generating a copy of the data base was not feasible. Furthermore, testing different modeling approaches online using the SIG was not recommended by Recorded Future. Another reason is that only a fixed time frame was considered, mostly affecting the number of references considered. Lastly the k-neighborhood graphs did not include all entities resulting in not including all

available entities on the SIG.

An alternative modeling approach would be to instead do a regression model on the risk score, instead of classifying Internet domains as either benign or malicious (or with the multi-class formulation). However the simplification makes the problem easier both from a modeling and an evaluation aspect. Moreover a regression model is not clearly preferable as the entities seem to cluster around certain risk scores, this clustering stems from risk rules. In addition the risk scores might not be accurate enough to motivate such a fine-grained classification.

1.4 Related Work

There are several examples of risk models in a cyber threat context in the literature. Wang et al [3] extends the Factor Analysis of Information Risk (FAIR) model with a Bayesian Network. Their work results in a probabilistic factor model which builds on a graph structure depending on the risk structure. Shin et al [4] use Bayesian networks for estimating cyber security risk for nuclear power plants, they also model the risk structure and use Bayesian networks to model risk on this structure. Compared to these approaches the approach we use mainly considers how entities relates to each other rather than the risk structure. The risk structure considered in the work by Wang et al and Shin et al describes how serious a threat is, for example with regard to severity of damage and how likely it is that the damage is made.

Carter et al [5] consider, among other applications, malicious domain analysis. They propose a probabilistic threat propagation where vertices have a probability of being of high risk depending on its neighbors risk probability. Their approach deals with the issue of exploding risk, an issue in several other probabilistic risk propagating methods.

Jacobs et al [6] suggest a logistic regression model on features for cyber vulnerabilities with a Common Vulnerability Enumeration (CVE). The features are publicly available information about the CVE:s such as vendors, tags and a reference counter. This approach is similar to our approach in this thesis with respect to the risk modelling. In Jacobs et al and in our project a support vector machine model is fitted on a feature space in order to model risk. They use categorical variables corresponding to vendors, tags etc for generating a feature space for classification.

Tavabi et al [7] builds a low dimensional feature representation, using neural networks, for CVE:s based on text from Dark Web Forums. This low dimensional feature space is then used for classifying CVE:s as either exploited or non-exploited with a radial basis function (RBF) support vector machine (svm). Notable is the good performance of their model they reach a F_1-score of 0.80. This approach is similar to our approach. We use information regarding how Internet Domains are mentioned and relate to other entities in order to build a feature representation, on which we use a RBF svm. One difference between this method and our is the data used. Recorded Future's database (SIG) contains Dark Web data but also several

other sources, such as Malware/Technical reporting. Hence more varying data is used compared to Tavabi et al.

Our models differ with respect to the graph representation of the data. In Carter et al [5], they use a graph structure but as a factor model rather than as a neighborhood graph model used for classification. While in Wang et al [3], Shin et al [4], Jacobs et al [6] and Tavabi et al [7] they do not use graph interpretations for the context of entities in order to perform risk estimations.

Xu et al [8] suggest a graph embedding using neural networks for, among other things, vulnerability detection. This approach is quite similar to our approach with regard to processing the data as a graph, mapping the graph data to a feature space, which is lastly used for classification. The approaches differ in methodology since our approach uses graph kernels, without neural networks, to generate feature representations. Then a more classical machine learning approach was used for classification, namely support vector machines. One reason for not using neural networks was because of lack of data with high risk. Another reason was that training neural networks is quite computationally demanding. However, as mentioned in [8], regarding runtime a neural network is superior to graph kernel approaches.

1.5 Overview

The thesis is divided into 6 chapters. The background in Chapter 2 contains graph definitions, a mathematical definition of the problem, an introduction to the method used including an example of the Weisfeiler-Lehman graph kernel. In addition, the evaluation metrics used are defined. In Chapter 3 the methodology is explained. Here the data collections steps, the feature generating steps and classification methods outlined. In Chapter 4 the results for the classification methods are described, additionally some data statistics are introduced. The results and method choices are discussed in Chapter 5. Lastly, these discussions are summarized in the Conclusion in Chapter 6.

2
Background

This section begins with definitions of graph terminology and problem. After this the classification methods used are introduced, followed by an introduction to graph kernels and the Weisfeiler-Lehman graph kernel framework. A quite extensive example then outlines the Weisfeiler-Lehman procedure, followed by an introduction to the Term Frequence times Inverse Document Frequency (tf-idf) transformer. Lastly, the evaluation metrics used are defined.

2.1 Graphs

A graph G is defined as a pair (V, E) where V is a set of vertices and E is a an edge set where $E = \{(v_i, v_j) =: e_{i,j} : v_i, v_j \in V, v_i \neq v_j\}$. A weighted graph has a weight edge function, $w : E \mapsto \mathbb{R}_{\geq 0}$. Only undirected graphs will be considered so the edges $e_{i,j} \in E$ and $e_{j,i} \in E$ are equivalent. Moreover a labeling function $l : V \mapsto \Sigma$ of the vertices to an alphabet Σ is considered. Σ is discrete in this thesis. The vertices also have a feature representation $f : V \mapsto \mathbb{N}^m$. The discrepancy between l and f is not obvious but is made for later convenience. The set of graphs as defined here is denoted as $\mathbb{G}$.

A *path* is an ordered set of vertices so that every adjacent pair is connected by an edge. Moreover a path is not a multiset, that is the same vertex can only appear once in the path. A *walk* is similar to a path with the exception that it is a multiset, that is the same vertex is allowed to appear several times. For an unweighted graph the shortest path between two vertices v_i and v_j corresponds to the path, or one of the paths, with lowest cardinality between v_i and v_j. If the graph is weighted the shortest path is defined as the path, or one of the paths, with the lowest corresponding weight sum. The degree of a vertex $d(v_i)$, $v_i \in V$, is a count of the number of adjacent vertices incident to v_i, explicitly $d(v_i) = |\{v_j : e_{i,j} \in E\}|$ where $|\cdot|$ denotes the cardinality of a set.

A closed 1-neighborhood for a vertex $v_i \in V$ is is defined in the following way,

$$N_1[v_i] = (V_{N_1[v_i]}, E_{N_1[v_i]})$$

where

$$V_{N_1[v_i]} = \{v_i\} \cup \{v_j \in V : (v_i, v_j) \in E\}$$

and

$$E_{N_1[v_i]} = \{e_{i,j} \in E : v_i, v_j \in V_{N_1[v_i]}\}.$$

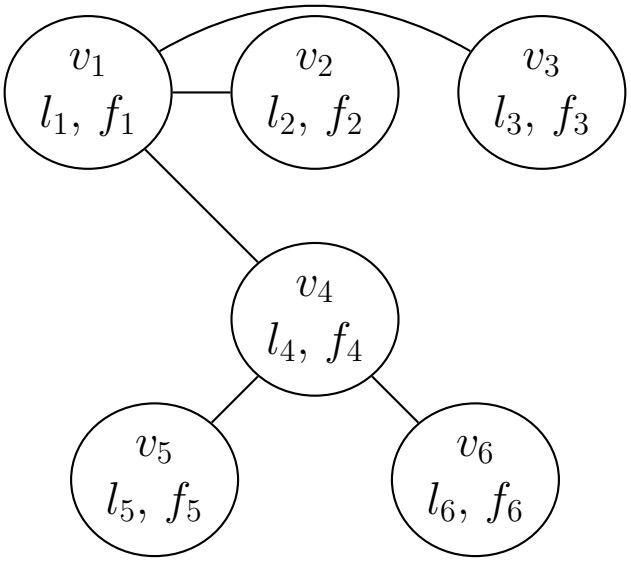


Figure 2.1: An example (undirected) graph. The vertices v_i have labels l_i and features f_i. The closed 1-neighborhood of vertex v_1 has vertex set $V_{N_1[v_i]} = \{v_1, v_2, v_3, v_4\}$ and edge set $E_{N_1[v_1]} = \{e_{1,2}, e_{1,3}, e_{1,4}\}$, while the closed 2-neighborhood of vertex v_1 is the whole graph. The open 1-neighborhood of vertex v_1 has vertex set $V_{N_1(v_i)} = \{v_2, v_3, v_4\}$, here v_1 is not present, while the edge set $E_{N_1(v_1)}$ is empty.

A closed 2-neighborhood for a vertex $v_i \in V$ is similarly defined as,

$$N_2[v_i] = (V_{N_2[v_i]}, E_{N_2[v_i]}) \tag{2.1}$$

where

$$V_{N_2[v_i]} = \{v_i\} \cup \{v_j \in V : (v_i, v_j) \in E\} \cup \{v_j \in V : (v_i, v_{j'}), (v_{j'}, v_j) \in E, v_{j'} \in V\}$$

and

$$E_{N_2[v_i]} = \{e_{i,j} \in E : v_i, v_j \in V_{N_2[v_i]}\}.$$

It is mostly closed 2-neighborhoods that will be considered further. Henceforth 2-neighborhood graphs will correspond to closed 2-neighborhood graphs, if open neighborhood graphs are considered this will be specified. An open 1-neighborhood for a vertex $v_i \in V$ is defined as,

$$N_1(v_i) = (V_{N_1(v_i)}, E_{N_1(v_i)})$$

where

$$V_{N_1(v_i)} = \{v_j \in V : (v_i, v_j) \in E\}$$

compare to a closed 1-neighborhood for a vertex v_i, here v_i is not included in the vertex set. The edge set is defined as follows,

$$E_{N_1(v_i)} = \{e_{i,j} \in E : v_i, v_j \in V_{N_1(v_i)}\}.$$

Note the difference in notation, an open 1-neighborhood uses $(\cdot)$ while a closed 1-neighborhood uses $[\cdot]$. These definitions can be generalized to open and closed k-neighborhood graphs, where k is a positive integer. Similarly as for 2-neighborhood graphs k-neighborhood graph will henceforth correspond to closed k-neighborhood graphs. An example graph is shown in Figure 2.1.

2.2 Problem definition

The objective is to learn a classifier of Internet domains as malicious or benign. We want to learn a classifiers $C(\cdot)$ of the closed k-neighborhood induced by the Internet domains, that is

$$l(v_i) = C(N[v_i]),$$

where v_i is an Internet Domain and l is a labeling function to the classes malicious and benign. In the multiclass formulation of the problem l instead maps vertices to $\{1, 2, \ldots, k\}$ where the labels corresponds to risk score intervals. The objective is to find a classifying function $C(N[v_i])$ which minimizes the misclassification error

$$E(C) = \sum_{i}^{n} L[l(v_i) - C(N[v_i])], \tag{2.2}$$

where L is a loss function.

2.3 Logistic Regression

A logistic regression model is a classifier which acts on a feature vector ϕ of inputs x_i with target values $y_i \in \{-1, 1\}$. The model with L_2 regularization corresponds to the optimization problem,

$$\min_{w} \ \|w\|_2^2 + \frac{\lambda}{n} \sum_{i=1}^{n} \ln\left(\exp\left(-y_i w^T \phi(x_i)\right) + 1\right) \tag{2.3}$$

where w is a weight vector, n is the number of data points, and λ is a hyperparameter[1] [11, p. 325-327]. The resulting classification function, or posterior distribution, becomes

$$p(1|x_i) = \frac{1}{1 + \exp\left(-w^T \phi(x_i)\right)}$$

and

$$p(-1|x_i) = 1 - p(1|x_i) = \frac{\exp\left(-w^T \phi(x_i)\right)}{1 + \exp\left(-w^T \phi(x_i)\right)}.$$

Moreover there is a multi class formulation where $y \in \{1, 2, \ldots, k\}$ this results in the following classification function

$$p(j|x_i) = \frac{\exp\left(w_j^T x_i\right)}{\sum_{l=1}^{k-1} \exp\left(w_l^T x_i\right)} \qquad \text{for } j = 1, 2, \ldots, k-1$$

[1] λ should be decided with cross validation.

and

$$p(k|x_i) = \frac{1}{\sum_{l=1}^{k-1} \exp\left(w_l^T x_i\right)}$$

where each class $j \in \{1, 2, \ldots, k-1\}$ have a corresponding weight vector w_j except the last class k which can be omitted without loss of generality [10, p. 197-198]. The squared error loss function for the multi-class classification (with L_2 regularization) is the following

$$\min_w \ \|w\|_2^2 + \frac{\lambda}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \qquad \text{where} \qquad \hat{y}_i = \underset{j\in\{1,2,\ldots,k\}}{\operatorname{argmax}} p(j|x_i). \tag{2.4}$$

Note that the error function is ordinal in the sense that misclassification loss is quadratic with respect to misclassification $|y_i - \hat{y}_i|$.

2.4 Support Vector Machines

There is an excellent introduction to support vector machines in Bishop: *Pattern Recognition and Machine Learning*, [10]. Do read that introduction if you are interested in support vector machines. With that said understanding support vector machines is important in order to follow the modelling approaches in this thesis, for that reason here comes a short introduction.

The objective of a two class classifier support vector machine (svm) is to linearly separate data with a hyper plane so that the maximum margin is obtained. This problem can be derived to the optimization problem,

$$\begin{aligned} &\textbf{minimize} && \|w\|_2^2 \\ &\textbf{subject to} && y_i(w^T\phi(x_i) - b) \geq 1 \ \ \forall\, i \in [0, 1, \ldots, n], \end{aligned}$$

where x_i are data points, ϕ is a mapping of the data, $y_i \in \{-1, 1\}$ are the corresponding labels and n is the number of data points. The weights w are the parameters to learn along with the bias term b. If the data is not perfectly linearly separable, this optimization problem is unfeasible. Reformulating it using slack variables is necessary in order to handle this case, the problem then becomes

$$\textbf{minimize} \qquad \|w\|_2^2 + \lambda\sum_{i=1}^{n} s_i \tag{2.5}$$

$$\begin{aligned} \textbf{subject to} \qquad & y_i(w^T\phi(x_i) - b) \geq 1 - s_i \ \ \forall\, i \in [0, 1, \ldots, n], \\ & s_i \geq 0 \ \ \forall\, i, \end{aligned} \tag{2.6}$$

where s_i are the slack variables. s_i larger than zero corresponds to breaking of the linear separation and is penalized with a linear factor λs_i for each s_i, λ is a hyperparameter[1]. This reformulation is called a soft margin svm.

In some situations the mapping ϕ might not be easily accessible, it might even exist in an infinite vector space such as for radial basis function (RBF), see Section

2.5. However we can derive the problem so that ϕ only appears in dot products, which deals with this issue. The problem is derived to the following form by using Lagrangian multipliers, the Karush Kuhn Tucker (KKT) conditions and the dual formulation,

$$\begin{aligned} &\textbf{minimize} && \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \phi(x_i)^T \phi(x_j) && (2.7) \\ &\textbf{subject to} && 0 \leq \alpha_i \leq \lambda \ \ \forall i \in [0, 1, \ldots, n], \\ & && \sum_{i=1}^{n} y_i \alpha_i = 0, \end{aligned}$$

where α_i are the Lagrangian multipliers corresponding to constraints (2.6) and λ is the constant in the objective function (2.5). Note here that $\phi(\cdot)$ only appears as a dot product. We define this dot product as a kernel function,

$$k(x_i, x_j) := \phi(x_i)^T \phi(x_j). \tag{2.8}$$

Only considering the kernel function and not ϕ is famously called the kernel trick.

We introduce the Gram matrix $K \in \mathbb{R}^{n \times n}$, where $k_{i,j} = k(x_i, x_j) \in K$. Then we can rewrite the objective function (2.7) in the following way,

$$\sum_{i=1}^{n} \alpha_i - \frac{1}{2} (\alpha \circ y)^T K (\alpha \circ y), \tag{2.9}$$

where $\circ$ denotes the Hadamard product. We see that the dual problem is well defined if K is positive semi-definite, that is if $\xi^T K \xi \geq 0$ for every $\xi \in \mathbb{R}^n$, by substituting (2.7) with (2.9) [12]. Note that the objective function (2.9) corresponds to the general loss function (2.2).

2.5 Radial basis function kernel

A commonly used kernel is the radial basis function kernel (RBF kernel). It is defined in the following way

$$k_{RBF}(x_i, x_j) := \exp\left(\frac{-\|x_i - x_j\|_2^2}{2\sigma^2}\right) \tag{2.10}$$

where $x_i, x_j \in \mathbb{R}^m$ and σ^2 is a hyperparameter. Note here that the corresponding ϕ function would be the infinite Taylor series of the exponential function [11, p. 110].

2.6 Graph Kernels

The classification is partly done over a graph structure G_i. Graphs are not directly well represented in a vector space. This motivates the subject of graph kernels. A graph kernel can either be explicit or implicit. An explicit graph kernel maps a

graph to a vector space and hence is connected to a vector representation $\phi : \mathbb{G} \mapsto \mathbb{R}^m$. An implicit graph kernel compares graph pairs without providing a vector representation. Compare this to the introduction of kernels in (2.8). In this case there is no accessible $\phi(G)$ but only a kernel over a pair of graphs [13],

$$k : \mathbb{G} \times \mathbb{G} \mapsto \mathbb{R}_{\geq 0}. \tag{2.11}$$

2.6.1 Vertex kernel

The vertex kernel is simply a counter of label occurrences in a graph. Since the vertex kernel is an explicit kernel we will define it with its feature vector ϕ_{vertex}. We consider a set of graphs

$$\mathcal{G} := \{G_1, G_2, \ldots, G_N\}, \tag{2.12}$$

where each graph $G_i \in \mathcal{G}$ has a corresponding alphabet (label set) Σ_i. We define the union of all alphabets

$$\Sigma := \bigcup_{i=1}^{N} \Sigma_i = \{\sigma_1, \sigma_2, \ldots, \sigma_k\} \tag{2.13}$$

and define the vertex label counting function

$$\#_l : \mathcal{G} \times \Sigma \mapsto \mathbb{N}_0 \tag{2.14}$$

where $\#_l$ counts the occurrence of the letter $\sigma_l \in \Sigma$ in a graph $G_i \in \mathcal{G}$. We consider such a function for all letters $\sigma_l \in \Sigma$ and introduce the feature vector

$$\phi_{vertex}(G_i) := \Big(\#_1(G_i, \Sigma), \#_2(G_i, \Sigma), \ldots, \#_k(G_i, \Sigma)\Big). \tag{2.15}$$

The vertex kernel, for two graphs $G_i, G_j \in \mathcal{G}$, becomes

$$k_{vertex}(G_i, G_j) := \phi_{vertex}(G_i)^T \phi_{vertex}(G_j).$$

2.6.2 Edge kernel

The edge kernel is slightly more involved than the vertex kernel. Consider again the graph set $\mathcal{G}$ as defined in (2.12) and the alphabet Σ as defined in (2.13). We define an edge label counter in the following way

$$\#_{(l,l')} : \mathcal{G} \times (\Sigma \times \Sigma) \mapsto \mathbb{N}_0 \tag{2.16}$$

where $\#_{l,l'}$ counts the number of edges incident to vertices with labels σ_l and $\sigma_{l'} \in \Sigma$ in a graph. We consider the labeled pairs as ordered pairs, for example if $(\sigma_l, \sigma_{l'})$ is ordered then we will order $(\sigma_{l'}, \sigma_l)$ before counting it. We define an edge label counter for all ordered pairs $(\sigma_l, \sigma_{l'}) \in \Sigma \times \Sigma$ and introduce the following feature vector,

$$\begin{aligned}\phi_{edge}(G_i) := \Big(&\#_{(1,1)}(G_i), \#_{(1,2)}(G_i), \ldots, \#_{(1,k)}(G_i), \\ &\#_{(2,2)}(G_i), \#_{(2,3)}(G_i), \ldots, \#_{(k,k)}(G_i)\Big)\end{aligned} \tag{2.17}$$

here $\Sigma \times \Sigma$ is omitted from the function argument for readability. Compared to (2.15) this feature representation is in a higher dimension, namely in dimension

$$\sum_{i=1}^{k}(k+1-i) = \sum_{i=1}^{k} i = \frac{k(k+1)}{2}.$$

The edge kernel then gets defined as

$$k_{edge}(G_i, G_j) := \phi_{edge}(G_i)^T \phi_{edge}(G_j)$$

for two graphs $G_i, G_j \in \mathcal{G}$. There is a version of the edge kernel where edge weights are considered as well,

$$\begin{aligned}\phi_{edge,w}(G_i) := \Big(&\sum_{(1,1)_j \in E_i} w_i((1,1)_j), \sum_{(1,2)_j \in E_i} w_i((1,2)_j), \ldots, \sum_{(1,k)_j \in E_i} w_i((1,k)_j), \\ &\sum_{(2,2)_j \in E_i} w_i((2,2)_j), \sum_{(2,3)_j \in E_i} w_i((2,3)_j), \ldots, \\ &\sum_{(k,k)_j \in E_i} w_i((k,k)_j)\Big) \quad (2.18)\end{aligned}$$

where $w_i((1,1)_j)$ denotes the weight of the j^{th} edge, labeled $(1,1)$ in the edge set E_i of graph G_i [14].

2.6.3 Weisfeiler-Lehman kernel

The Weisfeiler-Lehman kernel framework is introduced in Shervashidze et al: *Weisfeiler-Lehman Graph Kernel*, [14]. The kernels most relevant for this thesis within the Weisfeiler-Lehman kernel framework are the vertex histogram kernel, or *subtree* kernel, and the edge histogram kernel. Both of these kernels are explicit kernels [13]. The Weisfeiler-Lehman kernel framework is built on the Weisfeiler-Lehman isomorphism test. This test is performed with a relabeling procedure, see Figures 2.2 and 2.3 for two examples. An important property of the relabeling procedure is that the relabeling function $l_{re} : V \mapsto \mathbb{N}$ must be global in the sense that all graphs must use the same relabeling function. The suggested implementation of l_{re} is with a hashmap and a counter so that every considered relabeling can be found in constant time and every new label can be decided with the counter. When a new label, that is a label which have not already been relabeled, is relabeled the counter increases with one. The procedure is described in Algorithm 1. The Weisfeiler-Lehman relabeling procedure introduced in [14] is shown in Algorithm 2.

The Algorithm 2 considers a graph G and returns a set of relabeled graphs $\{G^0, G^1, \ldots, G^h\}$. When two graphs G_i and G_j are compared with any graph kernel k, see (2.11), within the Weisfeiler-Lehman kernel framework the Weisfeiler-Lehman kernel becomes the sum of the kernels for each relabeling iteration,

$$k^{WL}(G_i, G_j) := k(G_i^0, G_j^0) + k(G_i^1, G_j^1) + \cdots + k(G_i^h, G_i^h). \quad (2.19)$$

A more general formulation is with weights $\alpha_r \geq 0$ for all iterations $r \in \{1, 2, \ldots, h\}$ so

$$k^{WL,\alpha}(G_i, G_j) := \alpha_1 k(G_i^0, G_j^0) + \alpha_2 k(G_i^1, G_j^1) + \cdots + \alpha_h k(G_i^h, G_i^h).$$

As long as the kernel k yields a valid gram matrix, i.e a positive semi-definite matrix, the Weisfeiler-Lehman kernel also yields a positive gram matrix, for a proof see [14]. If the kernel k is explicit it has a corresponding feature vector ϕ, we introduce the concatenated feature vector

$$\phi^{WL}(G_i) := (\phi(G_i^0), \phi(G_i^1), \ldots, \phi(G_i^h)). \tag{2.20}$$

Algorithm 1 Relabeling function l_{re}

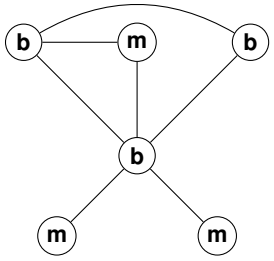
Input: label l_0
if l_{re} not initialized **then**
  l_{re} := empty `hashmap`
  $counter := 0$
end if
if $l_0 \in l_{re}$ **then**
  $l_0 = l_{re}(l_0)$
  return l_0
else
  $l_{re}(l_0) = counter$
  $l_0 = l_{re}(l_0)$
  $counter = counter + 1$
  return l_0
end if

Algorithm 2 Weisfeiler-Lehman relabeling iterations

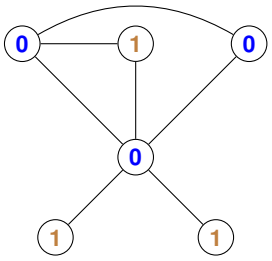
Input: h := number iterations
Input: $G = (V, E)$ with label function l
$G^0 := (V^0, E) = (\varnothing, E)$
for all $v_i \in G$ **do**
  Relabel v_i with l_{re} so that $l_i^0 := l_{re}(l(v_i))$
  Add v_i with label l_i^0 to V^0, i.e. $l^0(v_i) = l_i^0$
end for
for $j \in \{1, 2, \ldots, h\}$ **do**
  $G^j := (V^j, E) = (\varnothing, E)$
  for all $v_i \in V^{j-1}$ **do**
    Generate `label multiset`$(v_i) := \{\{l^{j-1}(v_j) : v_j \in V^{j-1}_{N_1(v_i)}\}\}$
    Sort `label multiset`(v_i) in ascending order using `counting sort`
    Add $l_{re}(v_i)$ as a prefix to `label multiset`(v_i) and convert to a string s_i
    Relabel v_i with $l_i^j := l_{re}(s_i)$
    Add v_i and l_i^j to V^j, i.e. $l^j(v_i) = l_i^j$
  end for
end for
return $\{G^0, G^1, \ldots, G^h\}$

2.6.4 Relabeling example

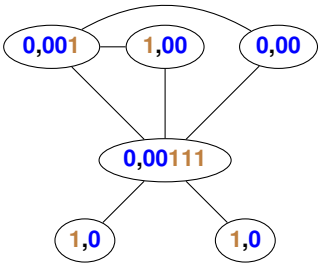
Two examples of the relabeling procedure from Algorithms 1 and 2 are shown in Figures 2.2 and 2.3. The graphs G_1 and G_2 are shown in their initial states in Subfigures 2.2a and 2.3a, receptively. The relabeling procedure is shown for both graphs in subsequent subfigures for two iterations. A corresponding global relabeling



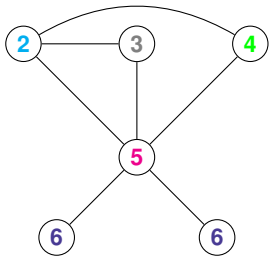
(a) Initial graph G_1: b denotes benign and m malicious.



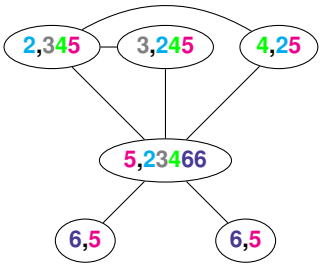
(b) Initial graph G_1^0: Relabeled with label function l_{re}, see **Relabel 1**.



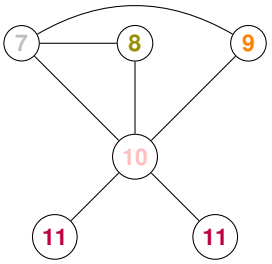
(c) First iteration: Relabeled with sorted neighboring labels.



(d) Graph G_1^1 relabeled with label function l_{re}, see **Relabel 2**. End of first iteration.



(e) Second iteration: Relabeled with sorted neighboring labels.



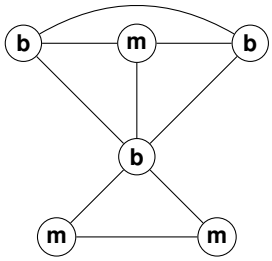
(f) Graph G_1^2 relabeled with label function l_{re}, see **Relabel 3**. End of second iteration.

Figure 2.2: Example graph G_1 with two Weifeiler-Lehman iterations. The relabeling steps can be found in A.1.

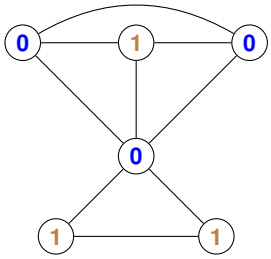
function as outlined in Algorithm 1 is shown in the Appendix, see A.1. In order to make the reasoning easier to follow there is a color encoding of the labels. The relabeling procedure results in the following relabeling function l_{re}:

$$
\begin{array}{rrrr}
b \mapsto 0 & m \mapsto 1 & \text{"}0,001\text{"} \mapsto 2 & \text{"}1,00\text{"} \mapsto 3 \\
\text{"}0,00\text{"} \mapsto 4 & \text{"}0,00111\text{"} \mapsto 5 & \text{"}1,0\text{"} \mapsto 6 & \text{"}2,234\text{"} \mapsto 7 \\
\text{"}3,245\text{"} \mapsto 8 & \text{"}4,25\text{"} \mapsto 9 & \text{"}5,23466\text{"} \mapsto 10 & \text{"}6,5\text{"} \mapsto 11 \\
\text{"}1,000\text{"} \mapsto 12 & \text{"}1,01\text{"} \mapsto 13 & \text{"}2,2512\text{"} \mapsto 14 & \text{"}12,225\text{"} \mapsto 15 \\
\text{"}5,22121313\text{"} \mapsto 16 & \text{"}13,513\text{"} \mapsto 17 & &
\end{array}
$$

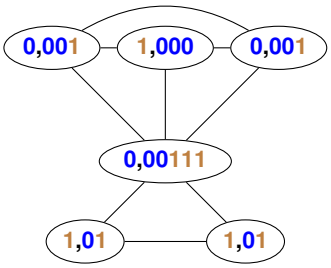
and the alphabet Σ is the set $\{0, 1, \ldots, 17\}$.



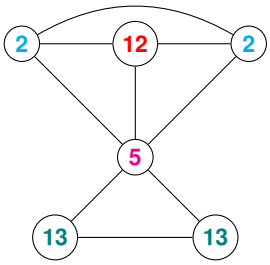
(a) Initial graph G_2: b denotes benign and m malicious.



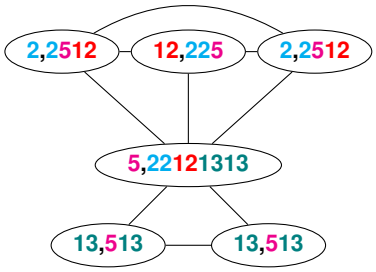
(b) Initial graph G_2^0: Relabeled with label function l_{re}, see **Relabel 4**.



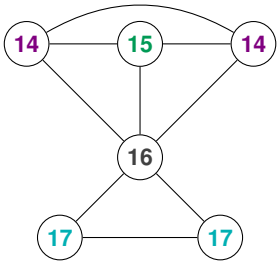
(c) First iteration: Relabeled with sorted neighboring labels.



(d) Graph G_2^1 relabeled with label function l_{re}, see **Relabel 5**. End of first iteration.



(e) Second iteration: Relabeled with sorted neighboring labels.



(f) Graph G_2^2 relabeled with label function l_{re}, see **Relabel 6**. End of second iteration.

Figure 2.3: Example graph G_2 with two Weifeiler-Lehman iterations. The relabeling steps can be found in A.1.

2.6.4.1 Vertex kernel

Using the vertex kernel in this frame work yields the following feature representation,

$$\phi^{WL}_{vertex}(G_i) = \Big(\underbrace{\#_0(G_i^0), \#_1(G_i^0),}_{0^{th}\ iteration}$$
$$\underbrace{\#_2(G_i^1), \#_3(G_i^1), \#_4(G_i^1), \#_5(G_i^1), \#_6(G_i^1), \#_{12}(G_i^1), \#_{13}(G_i^1),}_{1^{st}\ iteration}$$
$$\underbrace{\#_7(G_i^2), \#_8(G_i^2), \#_9(G_i^2), \#_{10}(G_i^2), \#_{11}(G_i^2), \#_{14}(G_i^2), \#_{15}(G_i^2), \#_{16}(G_i^2), \#_{17}(G_i^2)}_{2^{nd}\ iteration} \Big)$$

where $\#_k$ is the vertex label counter defined in (2.14), Σ is omitted from the function argument for readability. The feature representation ϕ_{vertex} is almost as defined in (2.15). The difference is that only elements which are nonzero in at least one of the feature vectors for graphs G_1 and G_2 are kept, for example since no vertices with label 0 are present in the first or second iteration the corresponding vertex counter $\#_0(G_i)$ is not included in the feature vector for those iterations. There are two reasons for this, firstly because it makes the vector easier to read and secondly because we do not want to add unnecessary feature dimensions to the classification problem.

For G_1 and G_2 as shown in Figures 2.2 and 2.3 the feature vectors with a vertex kernel becomes respectively

$$\phi^{WL}_{vertex}(G_1) = (\underbrace{3,3,}_{0^{th}\ iteration} \underbrace{1,1,1,1,2,0,0,}_{1^{st}\ iteration} \underbrace{1,1,1,1,2,0,0,0,0}_{2^{nd}\ iteration}),$$
$$\phi^{WL}_{vertex}(G_2) = (\underbrace{3,3,}_{0^{th}\ iteration} \underbrace{2,0,0,1,0,1,2,}_{1^{st}\ iteration} \underbrace{0,0,0,0,0,2,1,1,2}_{2^{nd}\ iteration}).$$

Note that the graphs G_1 and G_2 are quite similar, the vertex sets are exactly the same and there are only two more edges in G_2 compared to G_1. For the 0^{th} iteration the feature vectors are identical, for the first iteration the features are quite similar while for the second iteration they are completely different. The vertex kernel of the two graphs becomes

$$k^{WL}_{vertex}(G_1, G_2) = \phi^{WL}_{vertex}(G_1)^T \phi^{WL}_{vertex}(G_2) = 21.$$

2.6.4.2 Edge kernel

The edge kernel gives the following feature representation,

$$\phi^{WL}_{edge}(G_i) = \Big(\#_{(0,0)}(G_i^0), \#_{(0,1)}(G_i^0), \#_{(1,1)}(G_i^0) \tag{2.21a}$$
$$\#_{(2,2)}(G_i^1), \#_{(2,3)}(G_i^1), \#_{(2,4)}(G_i^1), \#_{(2,5)}(G_i^1), \#_{(2,12)}(G_i^1), \tag{2.21b}$$
$$\#_{(3,5)}(G_i^1), \#_{(4,5)}(G_i^1), \#_{(5,6)}(G_i^1), \#_{(5,12)}(G_i^1), \#_{(5,13)}(G_i^1), \tag{2.21c}$$
$$\#_{(13,13)}(G_i^1), \tag{2.21d}$$
$$\#_{(7,8)}(G_i^2), \#_{(7,9)}(G_i^2), \#_{(7,10)}(G_i^2), \#_{(8,10)}(G_i^2), \#_{(9,10)}(G_i^2), \tag{2.21e}$$
$$\#_{(10,11)}(G_i^2), \#_{(14,14)}(G_i^2), \#_{(14,15)}(G_i^2), \#_{(14,16)}(G_i^2), \tag{2.21f}$$
$$\#_{(15,16)}(G_i^2), \#_{(16,17)}(G_i^2), \#_{(17,17)}(G_i^2) \Big) \tag{2.21g}$$

where $\#_{(l,l')}$ is the edge label counter defined in (2.16). The feature representation is also here written without the possible edge pairs in $\Sigma \times \Sigma$ which do not appear in either G_1 nor in G_2, compare to (2.17). Note also that (2.21a) corresponds to the 0^{th} Weisfeiler-Lehman iteration, (2.21b) to (2.21d) corresponds to the first Weisfeiler-Lehman iteration and (2.21e) to (2.21g) corresponds to the second Weisfeiler-Lehman iteration. For G_1 and G_2 the corresponding feature vectors become the following,

$$\phi^{WL}_{edge}(G_1) = (\underbrace{1,3,4,}_{0^{th}\,iteration}\ \underbrace{0,1,1,1,0,1,1,2,0,0,0,}_{1^{st}\,iteration}\ \underbrace{1,1,1,1,1,2,0,0,0,0,0,0}_{2^{nd}\,iteration}),$$
$$\phi^{WL}_{edge}(G_2) = (\underbrace{2,3,5}_{0^{th}\,iteration}\ \underbrace{1,0,0,2,2,0,0,0,1,2,1,}_{1^{st}\,iteration}\ \underbrace{0,0,0,0,0,0,1,2,2,1,2,1}_{2^{nd}\,iteration}),$$

where the order in the vectors is the same as in (2.21). The edge kernel of G_1 and G_2 becomes

$$k^{WL}_{edge}(G_1, G_2) = \phi^{WL}_{edge}(G_1)^T \phi^{WL}_{edge}(G_2) = 33.$$

2.7 tf-idf

Term Frequence times Inverse Document Frequency (tf-idf) is a representation used for text data, a definition can be found in [17]. There is a natural interpretation of graphs as documents and the feature vectors, as defined in (2.20), as term counts. Let us define tf-idf for this interpretation instead. We consider a set of graphs $\mathcal{G}$, as in (2.12), with cardinality N. For a graph $G_j \in \mathcal{G}$ with feature vector $\phi^{WL}(G_j) \in \mathbb{R}^M$ define

$$TF_{ij} := \frac{\phi^{WL}(G_j)_i}{\max_k \phi^{WL}(G_j)_k}$$

where $\phi^{WL}(G_j)_i$ is the i^{th} element in $\phi^{WL}(G_j)$ and $\max_k \phi^{WL}(G_j)_k$ is the largest element. We define

$$IDF_i := \ln\left(N/n_i\right)$$

where n_i is the number of graphs, in $\mathcal{G}$, where the term $\phi^{WL}(\cdot)_i$ occurs. Lastly, the tf-idf transform becomes

$$tf\text{-}idf(G_j) := \left(TF_{ij} \cdot IDF_i\right)_i \text{ for } i = 1, 2, \ldots, M.$$

2.8 Evaluation metrics

In order to evaluate results the metrics *accuracy*, *error rate*, *precision*, *recall* and F_1 *score* are used. Let us initially, for the binary classification problem of vertices to the labels benign or malicious, define the concepts of *true positives* (TP), *false positives* (FP), *false negatives* (FN) and *true negatives* (TN) in the context of this thesis:

> TP := The number of malicious vertices which get an accurate estimation as malicious from the model.

FP := The number of benign vertices which get an inaccurate estimation as malicious from the model.

FN := The number of malicious vertices which get an inaccurate estimation as benign from the model.

TN := The number of benign vertices which get an accurate estimation as benign from the model.

These definitions can be expressed in a *confusion matrix*, see Table 2.1 for $k = 3$.

		Correct	
		Malicious	Benign
Estimated	Malicious	TP	FP
	Benign	FN	TN

Table 2.1: Confusion matrix for binary classification.

For the multiclass classification problem, the concepts of TP, FP, FN and TN gets slightly more involved. We define TP_i, FP_i, FN_i and TN_i for an arbitrary label $i \in \{1, 2, \ldots, k\}$:

TP_i := The number of vertices with true label i which get an accurate label i from the model.

FP_i := The number of vertices with true label $j \neq i$ which get an inaccurate label i from the model.

FN_i := The number of vertices with true label i which get an inaccurate label $j \neq i$ from the model.

TN_i := The number of vertices with true label j which get an accurate (or inaccurate) label $j' \neq i$ from the model.

These definitions are shown in the multiclass confusion matrix in Table 2.2 for $k = 3$.

We can now define the metrics, both in the binary classification and the multiclass classification. In the multiclass classification setting we use macro averaging (average per class) metrics, as this choice does not favour large classes (the alternative would be to use micro averaging which favours large classes) [18]. The motivation for this is that there is no apparent reason for why the larger classes should be more important. We denote the multiclass metrics with a subscripted M, for example $accuracy_M$, while for the the binary case we omit the M.

		Correct		
	Label	1	2	3
Estimated	1	TP_1	FP_1 / FN_2	FP_1 / FN_3
	2	FP_2 / FN_1	TP_2	FP_2 / FN_3
	3	FP_3 / FN_1	FP_3 / FN_2	TP_3

Table 2.2: Confusion matrix for three class classification. The non diagonal elements have two different names depending on in which context they are interpreted. For example FP_1 / FN_2, which corresponds to entry $e_{1,2}$ in the confusion matrix, denotes false positives with respect to label 1, while with respect to label 2 it denotes false negatives. Moreover if a value occurs more than once in the confusion matrix those values should be summed up to get the true value, for example $FP_1 = e_{1,2}+e_{1,3}$. True negatives are omitted.

2.8.1 Accuracy

The accuracy is defined as the correct estimations of the model divided by all estimations, or explicitly for the binary classification problem,

$$accuracy := \frac{TP + TN}{TP + FP + FN + TN}.$$

Similarly, for the multiclass classification problem,

$$accuracy_M := \frac{1}{k}\sum_{i=1}^{k}\frac{TP_i + TN_i}{TP_i + FP_i + FN_i + TN_i}.$$

2.8.2 Error rate

The error rate is the proportion of the estimations which is wrongly classified, hence in the binary setting

$$error\,rate := 1 - accuracy = \frac{FP + FN}{TP + FP + FN + TN},$$

and for the for the multiclass classification problem,

$$error\,rate_M := 1 - accuracy_M = \frac{1}{k}\sum_{i=1}^{k}\frac{FP_i + FN_i}{TP_i + FP_i + FN_i + TN_i}.$$

2.8.3 Precision

Precision is a measure of how many of the vertices which are classified as malicious actually are malicious in the binary problem, explicitly

$$precision := \frac{TP}{TP + FP}.$$

For the multiclass problem,

$$precision_M := \frac{1}{k} \sum_{i=1}^{k} \frac{TP_i}{TP_i + FP_i}$$

2.8.4 Recall

Recall is defined in the following way for a binary classification problem,

$$recall := \frac{TP}{TP + FN}.$$

It can be seen as a measure of how well the model manages to find malicious vertices. While, in the multiclass problem it is defined as

$$recall_M := \frac{1}{k} \sum_{i=1}^{k} \frac{TP_i}{TP_i + FN_i}.$$

2.8.5 F_1 score

Both recall and precision are important metrics for the classification problem, F_1 score is the harmonic mean of those metrics. Namely, for the binary problem

$$F_1\ score := 2 \left(\frac{1}{precision} + \frac{1}{recall} \right)^{-1} = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

and for the multiclass problem

$$F_1\ score_M := 2 \left(\frac{1}{precision_M} + \frac{1}{recall_M} \right)^{-1}.$$

3
Methods

In this section the graph data is described both with regard to its structure and with regard to how it is sampled. Then our classification models are described and lastly a section about evaluation of these models.

The overall modeling approach is outlined in Figure 3.1. The graph interpretation is outlined in Section 3.1.1 and the data collection is described in Section 3.1.4. The graph kernel embeddings of the neighborhood graphs are outlined in Section 3.1.5. Furthermore, the classification methods are described in Section 3.2. Lastly, the evaluation methods are described in Section 3.3.

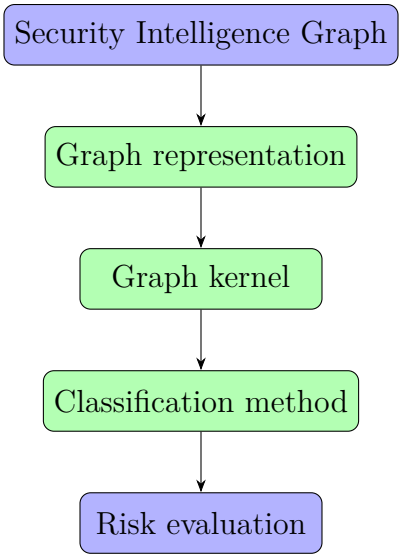


Figure 3.1: Schematic view of the modeling approach.

3.1 Data

Recorded Future's database the Security Intelligence Graph (SIG) is already a graph, it contains entities (vertices) and relationships both through references and with respect to ontology (edges). The graph representation derived from the SIG in our work is another graph representation, it considers less data and only the event graph, this representation is outlined below.

3.1.1 Data representation

The SIG was represented as several graphs $G^{MT} = (V, E^{MT})$, where E^{MT} corresponded to edges generated by text fragments of a media type MT and V corresponded to the entities on the SIG. Depending on how many entities a text fragment mentioned it gave rise to different amounts of edges, namely if it mentioned x number of entities it generated $\frac{1}{2}x(x-1)$ edges. Note that if a text fragment mentioned only one entity it did not generate an edge. Moreover note that it was only the edge sets that differed for the different graphs, the vertex sets stayed the same.

The text fragments are encapsulated in a data structure called *references* in Recorded Future's database, where information about the document the fragment comes from is stored as well. The information which was most prevalent in our modeling approach was time data and media type data. Some media types were excluded if they were either not informative or not prevalent enough in the data. In some cases several text fragment generated an edge labeled with the same media type, in this case a counter was updated which corresponds to a weight function w^{MT}. Since mostly there was not an obvious direction in the text fragments the graph is considered undirected.

Reference
Time data
Document
Media type
List of mentioned entities

Table 3.1: An overview of what is contained in a reference and used in the data collection.

3.1.2 Labels

Each entity which was considered has a risk score between 0 and 99. Instead of considering the risk scores the scores were translated to a labeling of entities as benign or malicious. Where entities with risk score of 80 or above were considered malicious and entities with risk score below 80 were considered benign. There were several reasons for this simplification. The risk scores below 80 are not supported by technical evidence of risk. Hence a relatively high risk score, but below 80, is not a confirmed risk but induced from circumstantial evidence. Risk scores above 80 are true positives in the sense that they are confirmed as having risk by technical evidence. Note here that true positives are with regard to a true labeling not necessarily captured in the considered data set. When evaluating models the labels in the considered data set were used as true labels since no other labels were accessible.

This labeling approach has issues with false negatives (again with respect to a true not accessible labeling) since some of the entities with only circumstantial evidence

of risk certainly are malicious. However one would not handle this issue with a continuous labeling using risk scores since some entities certainly have lower risk scores in the database than they should. In addition a two class problem is both easier to model and to evaluate.

A multiclass formulation of the classification problem was also implemented. The chosen risk scores intervals were

$$\{[0, 25), [25, 65), [65, 100)\}.$$

This choice was motivated by considering that a risk score in the interval [0, 25) is not an indication of risk (at least not confirmed risk). The choice of the mid interval [25, 65) was made since these risk scores indicate that there are some cyber risks regarding these Internet Domains, but not very hazardous. The choice of the interval [65,100) might be surprising since the threshold for the malicious label was 80. However a risk score above or equal 65 is considered high although there might not be any technical evidence confirming maliciousness. Moreover introducing a forth interval [65,80) decreased performance and hence instead considering a 3 class problem seemed reasonable. Lastly, having a threshold of 65 instead of 80 in the highest risk class made the problem more different compared to the binary classification problem. This choice therefore seemed like a more interesting problem to model.

3.1.3 k-neighborhoods

The choice made of closed neighborhood size was 2. Hence, $k = 2$ for the neighborhood graphs. This choice was made since only considering the 1-neighborhood of Internet Domains did not provide accurate risk evaluations. Moreover the 2-neighborhood graphs yielded much more accurate classification compared to only considering the 1-neighborhoods. Not increasing k to 3 was decided since the data collection step was quite time demanding and even $k = 2$ yielded quite large neighborhood graphs, see Figure 4.3. Additionally, considering the 3-neighborhood could yield too similar neighborhood graphs since the probability of different neighborhood graphs being connected increases with k.

3.1.4 Data collection

The data collection is outlined in a schematic view in Figure 3.2. For a more detailed description see Figure 3.3.

To access the data an application programming interface (API) was used. This API does not support extracting random subsets of data points. Moreover if a metric, i.e. the risk score, is specified in the query the entities with the highest available metric in the specified interval are returned. In order to get a more uniform distribution of risk scores for the two classes, benign and malicious, risk scores were sampled uniformly over the two defined risk score intervals. By implementing this method we did not sample the underlying distribution of risk scores in the SIG. If we would

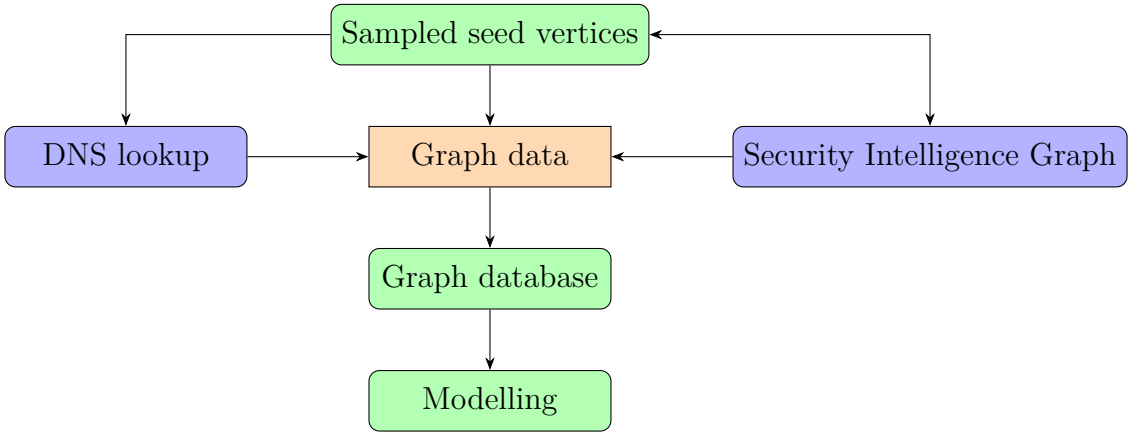


Figure 3.2: Schematic view of data collection.

have kept the risk score ratio of the SIG almost all risk scores would have been in the interval 0 to 5, see Figure 4.1.

The assumption that the 2-neighborhood of entities is important for the risk score made the proposed sampling preferable, since we wanted to sample each risk score interval so that we could learn the properties of the 2-neighborhoods for each interval. This can be seen in an active learning framework since we actively decide how the data is sampled. The resulting entities from the sampling was defined as seed vertices v_i^{seed}. These seed vertices were to be classified in the classification framework.

In order to generate the 2-neighborhood of the seed vertices several API calls must be made. Recorded Future's database was queried for references which mention any of the seed vertices. Since the amount of references was quite large, and hence not feasible to process, only references during the period from 2020-08-01 to 2020-11-17 were considered. This time interval can be justified by the fact that a risk evaluation is time dependent, observations that are made recently are quite a lot more important than observations made over 60 days ago. This is captured in the risk rules that Recorded Future uses in order to evaluate the risk score. Hence it is reasonable to not consider every reference that mentions the seed vertices, the time interval could however either be more conservative or liberal in size.

Entities that were mentioned in the reference, that were not among the seed vertices and were of a relevant type, that is either IP Address, Internet Domain, Malware, Threat Actor or Cyber Vulnerability, were stored for property queries. These property queries were needed for accessing properties of the vertices such as risk score. After property queries were performed for all stored entities the 1-neighborhood of the seed vertices in SIG were found. Next another reference search was performed to generate the 2-neighborhood of the seed vertex. The procedure with property queries was done and the whole 2-neighborhood was obtained.

The 2-neighborhood graphs were then enriched with Domain Name System (DNS) look up data, which was generated by querying another service provided by Recorded

Future. DNS look up can be done by using freely available services on the Internet, or from the terminal. This is however not advisable since the owner of the Domain can access which IP Address that asked for the DNS look up, something which might be hazardous for malicious Domains, since it might draw unwanted attention. The DNS look up was done by querying with Internet Domain names. Returned are other Internet Domain names and IP Addresses. Here a filtering step was done by only considering Internet Domains and IP addresses which were also found in the SIG. This was done since we want the properties of these entities as well. Internet Domains which were contained in the 2-neighborhood but not in the 1-neighborhood of any of the seed vertices were not queried for DNS information since that would have generated entities which were in the 3-neighborhood.

The data was stored in a graph database called Neo4j. As mentioned above media type or DNS information type generates an edge set. The vertices correspond to the entities. It was on this graph database data all modeling was performed.

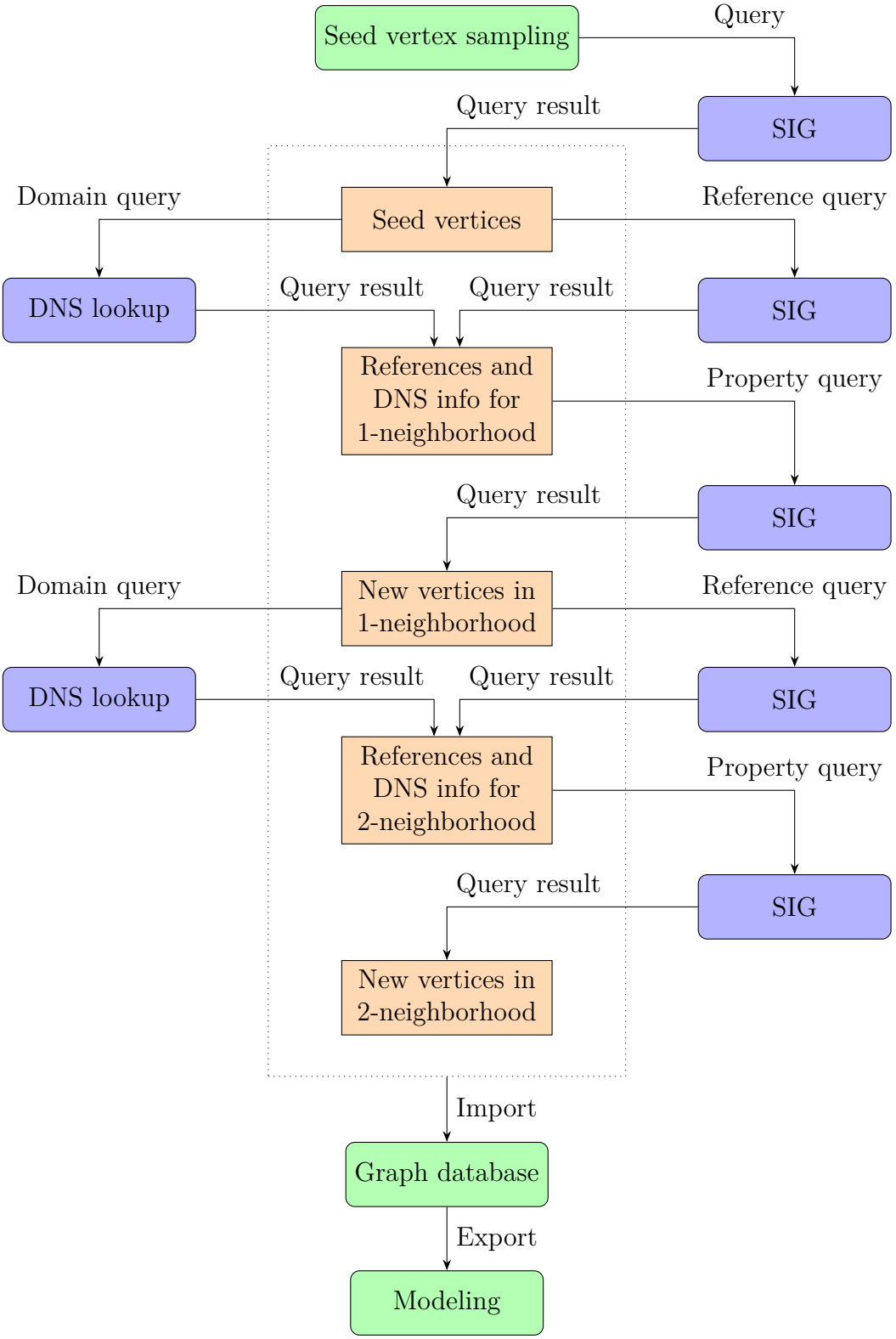


Figure 3.3: Breadth-first data collection procedure. The Security Intelligence Graph (SIG) corresponds to Recorded Future's database.

3.1.5 Feature representation

The feature representation was generated in Python. Firstly the Neo4j database got queried for seed vertices which generated a 2-neighborhood graph with the chosen edge label set, that is seed vertices which did not have any neighbors were filtered out.

The next step in the feature representation process was to generate the 2-neighborhood graphs for all seed vertices. As mentioned in Section 3.1.1 every edge set E^{MT} generated a graph of its own. This edge set was enriched with edges generated from DNS lookup information E^{DNS}, this information is verified and can be seen as a true relation hence including it in every edge set illustrates its importance. Moreover a lot of the sampled seed vertices would not have had a 2-neighborhood without these additional edges. We call this enriched edge set $E^{MT'}$ and the corresponding graph $G^{MT'}$. The 2-neighborhood graph of a seed vertex v_i^{seed} in this graph becomes, as defined in (2.1),

$$N_2^{MT'}[v_i^{seed}] = (V_{N_2[v_i^{seed}]}, E^{MT'}_{N_2[v_i^{seed}]})$$

where

$$E^{MT'}_{N_2[v_i^{seed}]} := E^{MT}_{N_2[v_i^{seed}]} \cup E^{DNS}_{N_2[v_i^{seed}]}$$

and if weights were considered $w^{MT'} := w^{MT} + w^{DNS}$.

$N_2^{MT'}[v_i^{seed}]$ for the seed vertices were put through the Weisfeiler-Lehman framework as described in Section 2.6.3. The number of Weisfeiler-Lehman iterations was set to two, this was motivated by considering that it is mainly two order and lower relationships that were captured in the 2-neighborhood graphs. As discussed in Section 2.6.3 the relabeling function l_{re} (see Algorithm 1) was the same for all graph data.

The initial vertex labeling was with respect to entity class along with a malicious or benign prefix, except for the seed vertex which got a seed prefix instead. If a seed vertex appeared in another seed vertex' 2-neighborhood graph, that vertex was labeled with a benign prefix. This was done since we wanted to be conservative in spreading risk, moreover giving it a seed prefix would have made different 2-neighborhood graphs more similar than necessary.

The initial vertex labeling in the multiclass classification model was done in a similar manner, expect the labeling was over risk score intervals $\{[0, 25), [25, 65), [65, 100)\} \mapsto \{1, 2, 3\}$, where the prefix was among labels $\{1, 2, 3\}$. Similarly if a seed vertex appeared in another seed vertex' 2-neighborhood graph, that vertex was labeled with a 1 prefix.

This process generated a feature representation ϕ^{WL}, as in (2.20), for every $N_2^{MT'}[v_i^{seed}]$. We concatenated the feature vectors and call the resulting matrix Φ. Note here that both the edge and vertex kernel were used, yielding Φ_{edge} and Φ_{vertex} respectively. The more general notation Φ is used for simplicity, since the same reasoning applies to both kernels.

A regularization was done by filtering out labels, the labels corresponds to the columns, which did not occur in more than 5 of the 2-neighborhood graphs in Φ. This was an effort to reduce the feature space and specifically handle the very large graphs which can be considered as outliers, see for example Figure 4.3. The choice of 5 may seem like an arbitrary limit, in some sense it is. The limit was chosen after looking at how the limit reduced the feature space of Φ. Ideally this limit should be chosen with cross validation, however doing this among all other model choices was too computationally demanding.

An other regularization was done by using the tf-idf transformer, see Section 2.7. This was done since comparing large graphs with small graphs is an issue within the Weisfeiler-Lehman framework. A large graph has a lot of vertices, hence the counters in the edge and vertex kernels will give high values. For smaller graphs these counters will give lower values. The issue arise when comparing graphs of different size, a comparison with a large graph will on average give higher similarity values, that is kernel values, even though the graphs might be quite different. Hence the regularization both with respect to graph size seemed reasonable. The other part of the tf-idf transformer is that common labels is given less importance, this also seemed reasonable to use.

3.2 Model

Two models were used for binary classification, namely logistic regression, see Section 2.3, and support vector machines with a RBF kernel, see Sections 2.4 and 2.5. In the binary case the labeling used was $\{\text{benign, malicious }\} \mapsto \{-1, 1\}$. For the multi-class problem multi-class logistic regression was used, see Section 2.3. The implementations of the models used are from the Python library scikit-learn [15].

3.2.1 Imbalanced data

The data was imbalanced both with respect to the binary labels and the multiclass labels, see Table 4.1. This lead to issues regarding misclassification loss, the larger class or classes became dominant in the objective functions, see (2.2), (2.3), (2.4) and (2.5). In order to handle this the loss function was balanced with respect to the labels proportions in the data. For example, in the binary logistic regression model the objective function as defined in (2.3) was

$$\|w\|_2^2 + \frac{\lambda}{n}\sum_{i=1}^{n} \ln\left(\exp\left(-y_i w^T \phi(x_i)\right) + 1\right).$$

In the balanced version the objective function instead becomes

$$\|w\|_2^2 + \frac{\lambda}{n}\sum_{i=1}^{n} p_{y_i} \ln\left(\exp\left(-y_i w^T \phi(x_i)\right) + 1\right), \text{ where } p_{y_i} := \frac{n}{n_{y_i} n_{label}}.$$

Here p_{y_i} is the inverse class proportion in the data, n is the number of data points, n_{y_i} is the number of data points with label y_i and n_{label} is the number of labels

considered, here either 2 or 3. The choice of including n_{label} was made since n_{label} is included in the definition of balanced accuracy [16]. Similarly, p_{y_i} was multiplied with the slack variables in (2.5).

3.2.2 Logistic regression

The logistic model was trained using cross validation with 5 five folds. The hyperparameter considered was λ in (2.3) and the parameter sweep was over the set $\{10^{-2}, 10^{-1}, 1, 10^1, 10^2, 10^3\}$. The optimization was performed with regard to the cross entropy loss function as defined in (2.3).

3.2.3 Multi-class logistic regression

In the multi-class formulation of logistic regression the labels were the risk score intervals

$$\{[0, 25), [25, 65), [65, 100)\} \mapsto \{1, 2, 3\}. \tag{3.1}$$

Cross validation, with 5 folds, was used for deciding λ in (2.4), also here the considered values were in $\{10^{-2}, 10^{-1}, 1, 10^1, 10^2, 10^3\}$. The training was optimized with respect to the function defined in (2.4) with $k = 3$.

3.2.4 Support vector machines with a RBF kernel

The features, which come from the Weisfeiler-Lehman (WL) graph kernels, were put through another kernel, namely the radial basis function (RBF) kernel. This approach was motivated by better metrics on the validation set (see Section 3.3) and by that the RBF kernel generally improved accuracy on WL feature data [13]. The hyperparameters to be decided were λ in (2.7) and σ^2 in (2.10). They were decided using cross validation with 5 folds over the parameter values

$$(\lambda, \sigma^2) \in \{10^{-2}, 10^{-1}, 1, 10^1, 10^2, 10^3\} \times \{10^{-2}, 10^{-1}, 1, 10^1, 10^2, 10^3\}.$$

The optimal pair (λ, σ^2) of the 36 possible pairs was chosen with respect to F_1 score, see Section 2.8.5.

3.3 Evaluation

The seed vertices were randomly split into a training and a test set, with 80 % and 20 % of the seed vertices respectively. There is a potential issue with this classical approach of the hold out method and that is that seed vertices from the training data might appear in the neighborhoods of the seed vertices in the test set. This is not necessarily a problem since we deal with graph data, which is connected, hence it is reasonable to assume that this case will also occur in a real-world scenario. However, since one can argue that sharing some parts of the graph in both the training and test sets is a case of data leakage a smaller test set was also used for evaluation. This smaller test set, X_{test}^{strict}, was a subset of seed vertices in the test set where no training seed vertices appeared in the neighborhoods. Hence two test sets

were used, the test set X_{test} and the strict test set $X_{test}^{strict} \subset X_{test}$ to evaluate models.

The main issue with X_{test}^{strict} was that its restriction drastically reduced the cardinality compared to the test set's X_{test} cardinality. This was especially the case regarding Internet Domains with high risk scores (≥ 80). Moreover one can argue that the Internet Domains in X_{test}^{strict} are outliers. For an illustration of the potential issues see Figure 3.4.

Throughout the modeling 20 % of the training data was kept as as a validation set. This was done in order to evaluate different modeling approaches, with regard deciding edge sets $E^{MT'}$, considering edge weights and the tf-idf transformer. However, when the final models were trained all training data was used.

3.3.1 Reference models

In order to evaluate the models trained on the Weisfeiler-Lehman data, reference models were trained as well. The motivation for this is that the data considered is not publicly available and hence there are no other clear benchmark models to compare with. The reference models were trained on features generated, without the Weisfeiler-Lehman framework, either with a vertex or an edge kernel. This corresponds to zero relabeling iterations in (2.19). The classification model used was support vector machine with a radial basis function kernel, see Section 3.2.4.

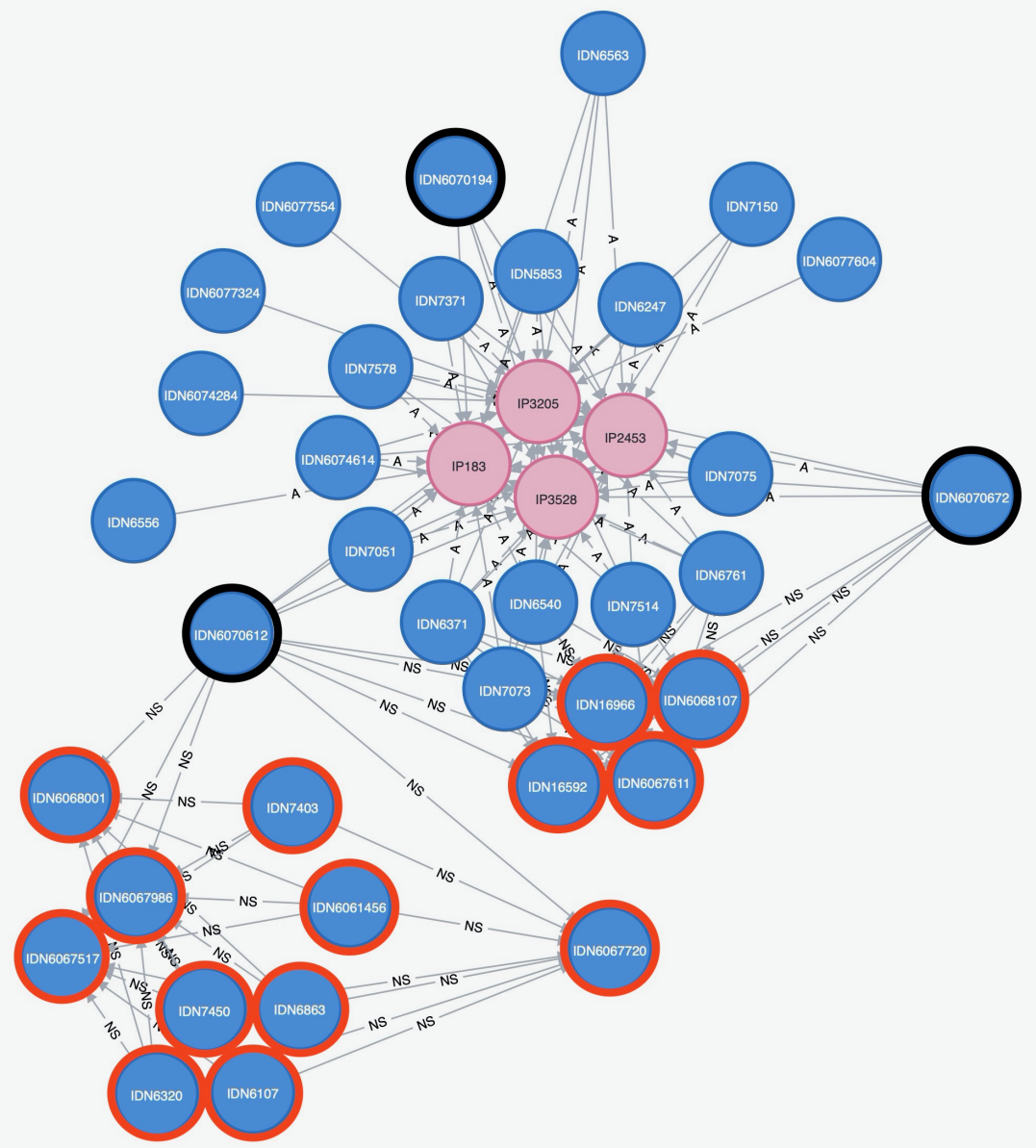


Figure 3.4: 2-neighborhood graphs, with edge sets generated only with DNS data, for seed vertices IDN6070194 and IDN6070612. The image was generated by Neo4j Desktop Browser. The blue vertices are Internet Domains and the pink vertices are IP addresses, the names are anonymized. Edge names in the graphs are either *NS*, which denotes name server, or *A* which corresponds to the IP address a Domain refers to. These 2-neighborhood graphs contains three seed vertices namely IDN6070194, IDN6070612 and IDN6070672 which are marked with black circles. Moreover the vertices marked with orange circles belong to the 2-neighborhood of seed vertex IDN6070612 but not to the 2-neighborhood of seed vertex IDN6070194. If IDN6070194 was in the training set and IDN6070612 was in the test set, IDN6070612 is not allowed to be in the strict test set because its 2-neighborhood graph contains IDN6070194. The 2-neighborhood graphs for respective seed vertex are quite different though, since the 2-neighborhood graph of seed vertex IDN6070612 is considerably larger.

4
Results

In this section data statistics can be found describing the data set used. Then results of risk models are shown.

4.1 Data statistics

The Security Intelligence Graph (SIG), which corresponds to Recorded Future's database, contains several entity types. Some of these categories were considered, namely IP Address, Internet Domain Name, Cyber Vulnerability and Malware. Members of the category Malware do not have risk scores as they are, per definition, harmful. The distribution of the other categories risk scores are outlined in Figure 4.1, where the y-axis is on logarithmic scale. Figure 4.1 describes how different risk score intervals relate in magnitude compared with each other for respective entity type. It is noticeable that the specific numbers are not visible and this is the case since we do not want to accurately describe the whole data base with consideration to Recorded Future.

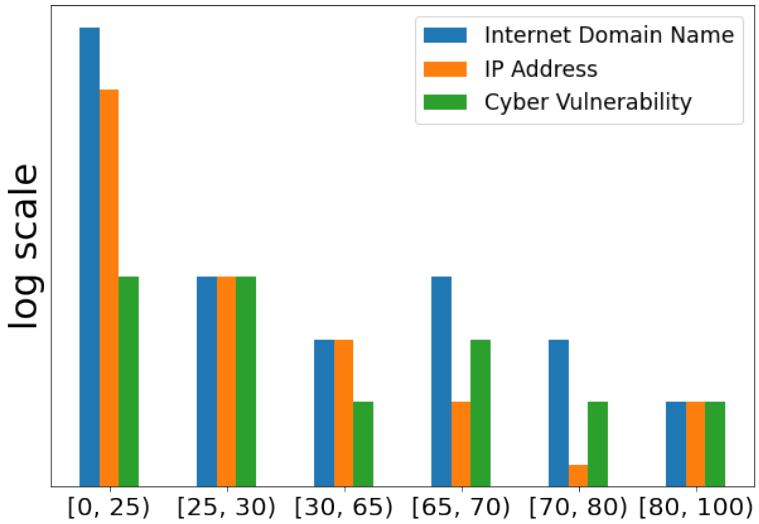


Figure 4.1: Distribution of entities' risk scores in the SIG (2020-12-08). The entity type Malware do not have risk scores. Note that the y-axis is on logarithmic scale and that the specific values are not described.

4.1.1 Data set statistics

As mentioned before a subset of the entities on SIG was used, the distribution of these entities are shown in Table 4.1.

Risk score interval and label	# Internet Domain Name	# IP Address	# Cyber Vulnerability	# Seed Internet Domain Name
1: $[0, 25)$	127 225	28 363	40	1 576
2: $[25, 65)$	1 322	5 231	63	764
3: $[65, 100)$	4 189	1 307	52	1 886
benign: $[0, 80)$	132 308	33 867	146	3 828
malicious: $[80, 100)$	428	1 034	9	398

Table 4.1: Distribution of entities' risk scores in the data set. The entity type Malware (199 such entities in the data set) do not have risk scores.

The different edge sets which were considered in the final models were generated either with Dark Web Market references enriched with DNS data, with Malware / Vulnerability Technical Reporting references enriched with DNS data or just with DNS data, see Section 3.1.5. A count of the number of edges for each edge set can be found in Table 4.2. The following abbreviations are used for this data:

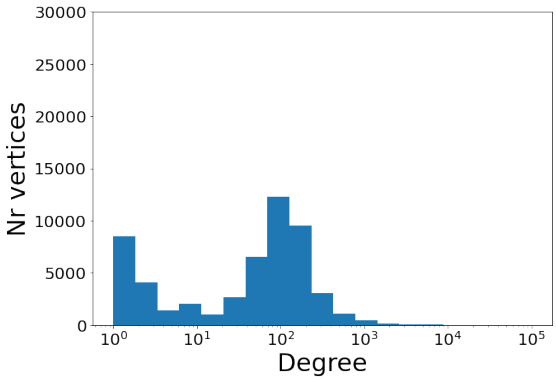
DWM′ := Dark Web Market references enriched with DNS data.
MVTR′ := Malware / Vulnerability Technical Reporting references enriched with DNS data.
DNS := Only DNS data.

Data	Number of edges
DWM′	3 328 247
MVTR′	1 982 420
DNS	22 762
Other	698 300

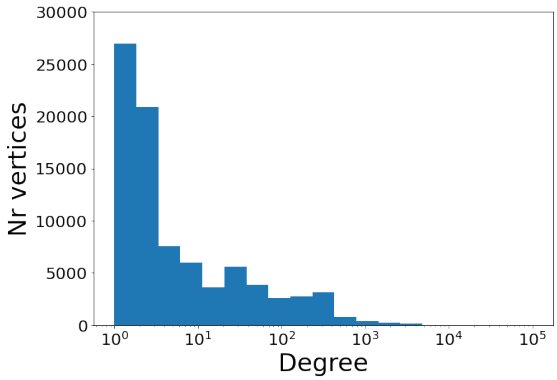
Table 4.2: Number of edges in the considered data set. Other denoted edges generated by other media type data than DWM and MVTR and not by DNS data. This is the total number of edge, while in subsequent statistics the effective number of edges is considered. For example, if an edge $e_{i,j}$ exists in both E^{DWM} and in E^{DNS} it is counted twice in $E^{DWM'}$ here, while in subsequent statistics it is counted once.

The degrees of vertices with edge sets generated by DWM′, MVTR′ and DNS have tailed distributions. They were also different between edge sets, see Figure 4.2. As expected the degree were lower generally lower for DNS, since this information were included in DWM′ and MVTR′. Note also considering DWM′ that more vertices had a higher degree, indicating that the Dark Web Market data mentioned more

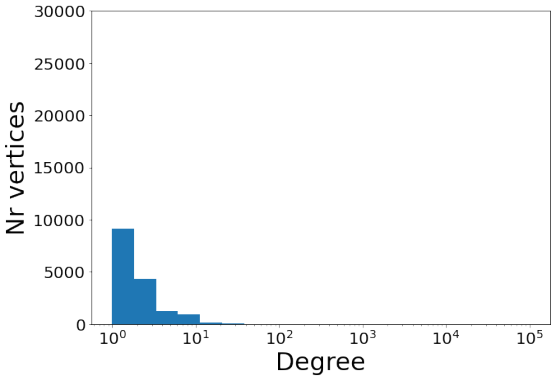
entities than the Malware / Vulnerability Technical Reporting data. Noticeable here is also that the number of vertices was larger when considering MVTR′. This is the case since DWM′ data lead to higher degrees while MVTR′ lead to a larger graph, with respect to vertex sets, with lower average degrees. Moreover as seen in Figure 4.2, the degree histogram for edge set DWM′ is bimodal whereas the others are unimodal.



(a) Degree histogram of vertices with edge set $E^{DWM'}$.



(b) Degree histogram of vertices with edge set $E^{MVTR'}$.

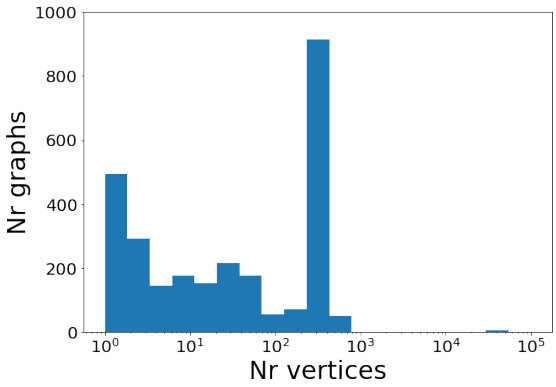


(c) Degree histogram of vertices with edge set E^{DNS}.

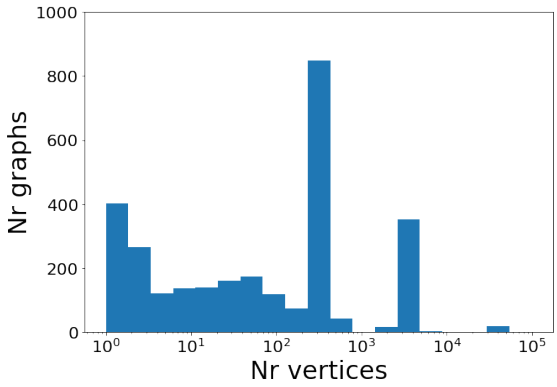
Figure 4.2: Degree distributions for edge sets generated by DWM′, MVTR′ and DNS, respectively. Remarkable is the second mode in the degree histogram for edge set DWM′.

4.1.2 2-neighborhood graph statistics

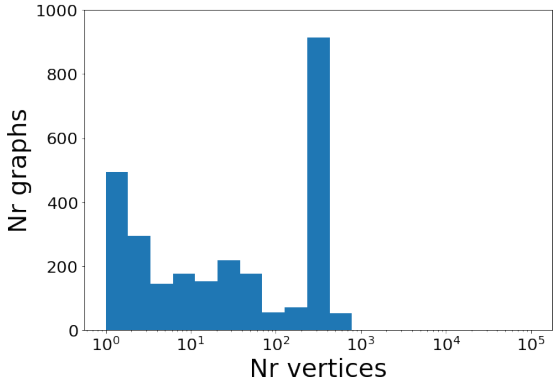
The 2-neighborhood graphs, which were put through the Weisfeiler Lehman framework, differ a lot in size, both with respect to edge sets and vertex sets. In Figure 4.3 the distributions of 2-neighborhood graphs sizes with respect to vertex counts are shown. Noticeable is that 2-neighborhood graphs with edge set $E^{MVTR'}$ were generally larger. Although, some very large 2-neighborhood graphs with edge set $E^{DWM'}$ were also present.



(a) With edge set $E^{DWM'}$.



(b) With edge set $E^{MVTR'}$.



(c) With edge set E^{DNS}.

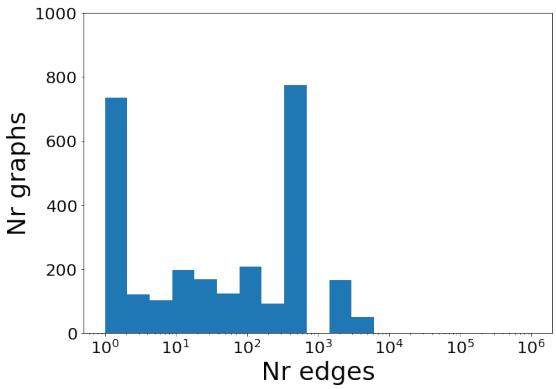
Figure 4.3: Vertex histograms of 2-neighborhood graphs with the edge sets generated by DWM′, MVTR′ and DNS. The high peaks around 500 vertices in all histograms seem untypical for graphs. This pattern probably arose from the fact that these graphs are 2-neighborhood graphs.

Regarding edges, the distributions of the 2-neighborhood graphs edge set's cardinality are shown in Figure 4.4. A similar pattern can be seen here as in Figure 4.3, the 2-neighborhood graphs with edge set $E^{MVTR'}$ had generally more edges compared to the other edge sets. We can also notice that $E^{DWM'}$ and E^{DNS} generally lead to quite similar edge sets.

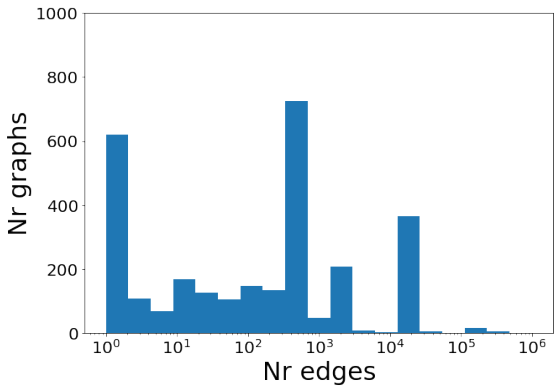
A pattern in both Figure 4.3 and Figure 4.4 is that there is a quite high peak around 500 vertices or edges. Degrees in real graphs generally follow a factor law, more similar to Figure 4.2 [19]. This one might expect would lead to graphs sizes, both with respect to vertex and edge sets, following a similar pattern. However since the 2-neighborhood graphs were connected, large clusters are likely to be included in many 2-neighborhood graphs. This likely explains the histograms.

Separating the histograms in Figure 4.3 and Figure 4.4 for benign and malicious seed nodes is done in Figures 4.5 and 4.6, respectively. Noticeable is that small 2-

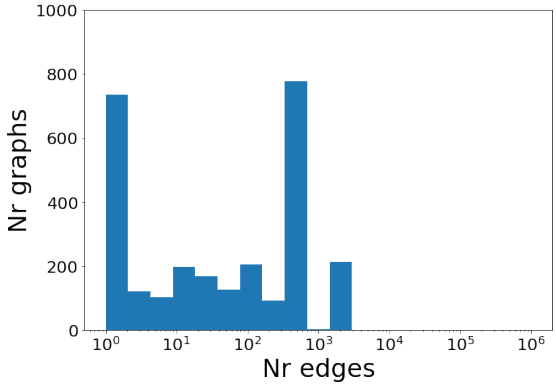
neighborhood graphs generally meant that the seed vertex was benign. This seems reasonable since this means that there was not a lot of information about such seed vertices. However it does not seem to be possible to simply use these histograms for classification. This motivates a more complex modeling.



(a) With edge set $E^{DWM'}$.

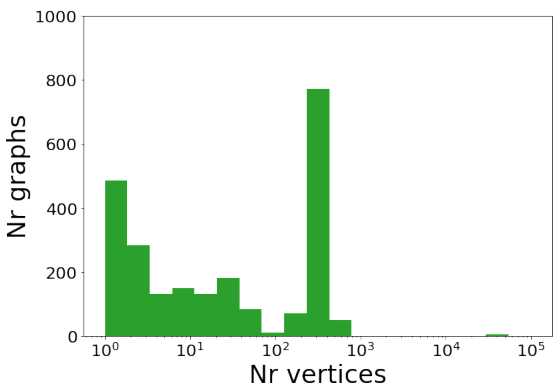


(b) With edge set $E^{MVTR'}$.

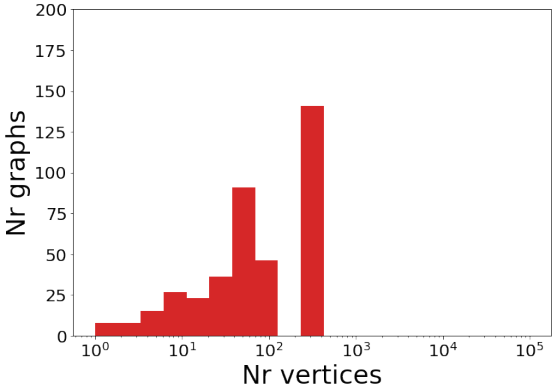


(c) With edge set E^{DNS}.

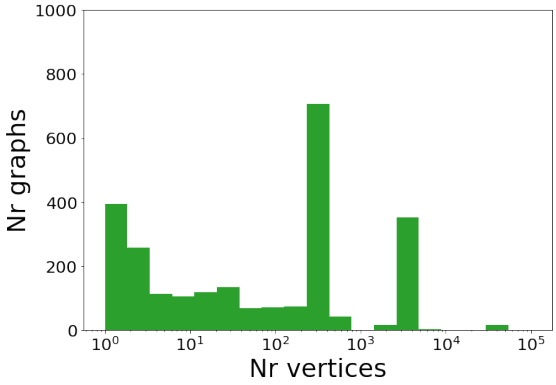
Figure 4.4: Edge histograms of 2-neighborhood graphs with the edge sets generated by DWM′, MVTR′ and DNS. The high peaks around 700 edges in all histograms seem unusual for graphs. This pattern probably stems from the fact that 2-neighborhood graphs are likely to share some vertices.



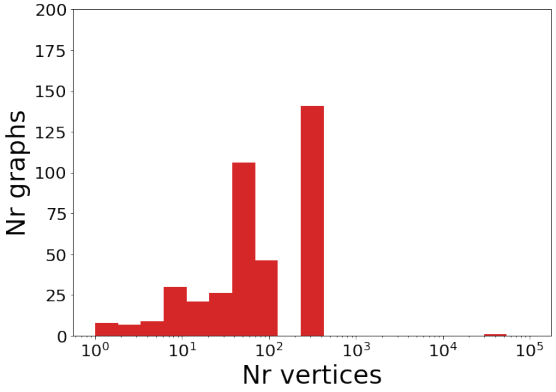
(a) Benign seed vertices with $E^{DWM'}$.



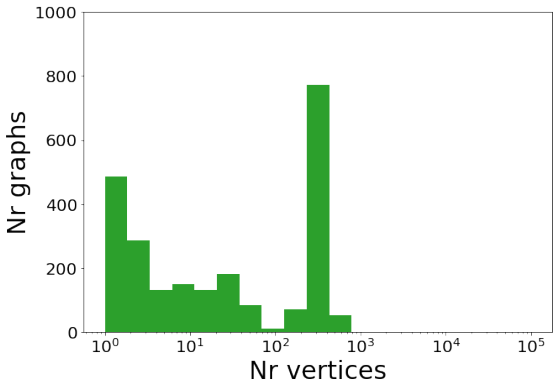
(b) Malicious seed vertices with $E^{DWM'}$.



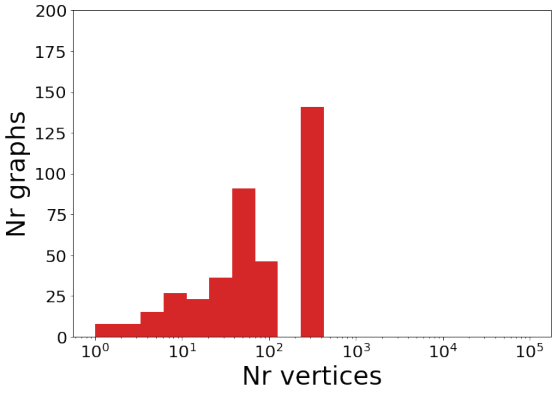
(c) Benign seed vertices with $E^{MVTR'}$.



(d) Malicious seed vertices with $E^{MVTR'}$.

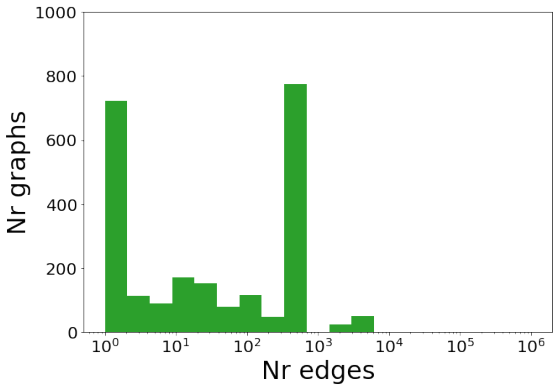


(e) Benign seed vertices with E^{DNS}.

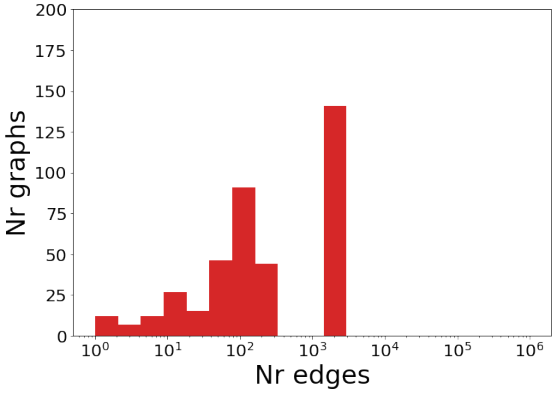


(f) Malicious seed vertices with E^{DNS}.

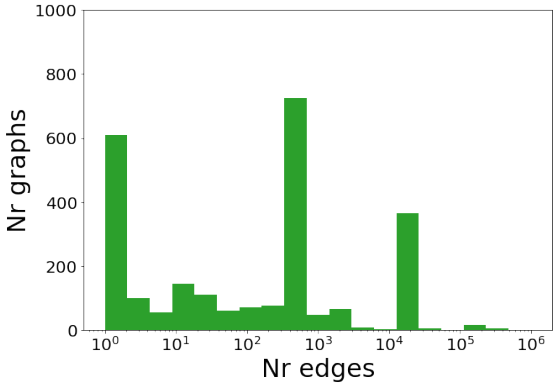
Figure 4.5: Vertex histograms of 2-neighborhood graphs with the edge sets generated by DWM′, MVTR′ and DNS. The histograms differ in shape between the two classes, although only using these histograms does not seem like a good choice in order to get an accurate classification.



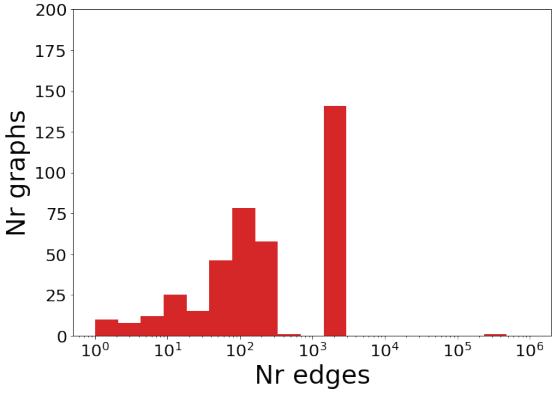
(a) Benign seed vertices with $E^{DWM'}$.



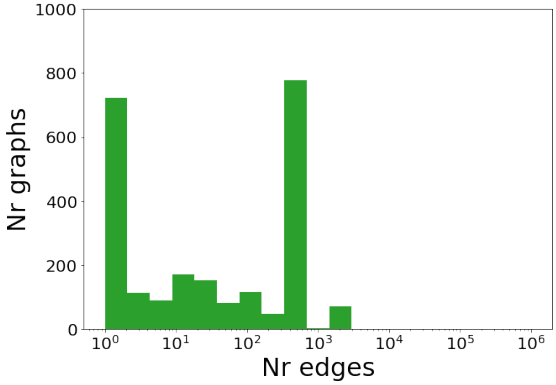
(b) Malicious seed vertices with $E^{DWM'}$.



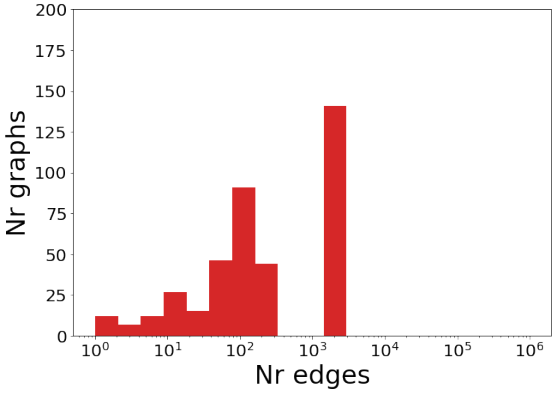
(c) Benign seed vertices with $E^{MVTR'}$.



(d) Malicious seed vertices with edge set $E^{MVTR'}$.



(e) Benign seed vertices with E^{DNS}.



(f) Malicious seed vertices with E^{DNS}.

Figure 4.6: Edge histograms of 2-neighborhood graphs with the edge sets generated by DWM′, MVTR′ and DNS. The distributions differ between the two classes, although only using these histograms does not seem feasible for an accurate classification.

4.2 Performances

There were two classification problems which were addressed. First, a binary classification problem of seed vertices to the classes benign and malicious. Second, a three class, multiclass, classification problem of seed vertices to the risk score intervals [0,25), [25,65) and [65,100). The models were trained and evaluated on graphs generated by different information, the abbreviations DWM′, MVTR′ and DNS denotes the different information categories, for a definition see Section 4.1.1.

4.2.1 Binary classification problem

The performances of the considered binary models with respect to accuracy and F_1 score are shown in Figure 4.7. The kernel used in the Weisfeiler Lehman framework was the vertex kernel and the number of relabeling iterations used was 2, let us denote this WL2. For the reference models the edge kernel was used without any relabeling iterations, this corresponds to zero Weisfeiler Lehman iterations. The results presented in Figure 4.7 are a subset of the results, for a more extensive display of the results see Appendix A.2. We see in Figure 4.7 that the performances of the reference models are quite similar to the WL2 models, both with respect to accuracy and F_1 score. We also see that only considering DNS data did not significantly reduce performance of the models.

The error rates of the considered models are displayed in Figure 4.8. In Subfigure 4.8a, the error rates are displayed for all test data, note that this information is contained in Figure 4.7 but here on a more readable scale. In Subfigure 4.8b the error rates are shown for the strict data set, see Section 3.3.

In Table A.1, in the Appendix, the results for logistic regression models are shown by features representation from WL2 and only an edge kernel. We can see here that that with a logistic regression classification method the WL2 models are significantly better than with an edge kernel. This suggests that the Weisfeiler-Lehman framework allows for linearly separable classes while an edge kernel does not. Note that the non-linear method radial basis function support vector machine with an edge kernel seems to work almost as well as the more complex WL2 models, suggesting that it is possible to find a non-linear decision boundary for the edge kernel.

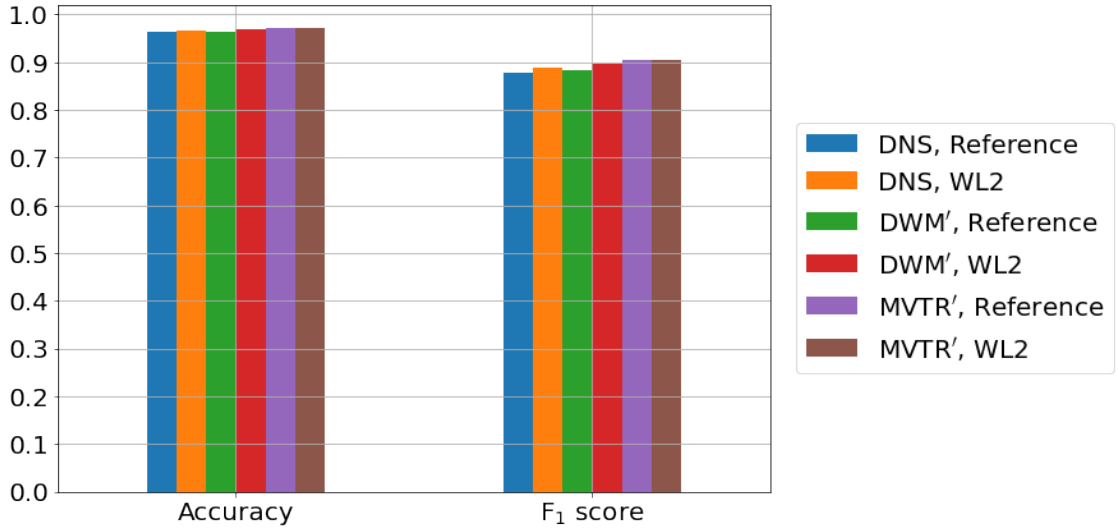
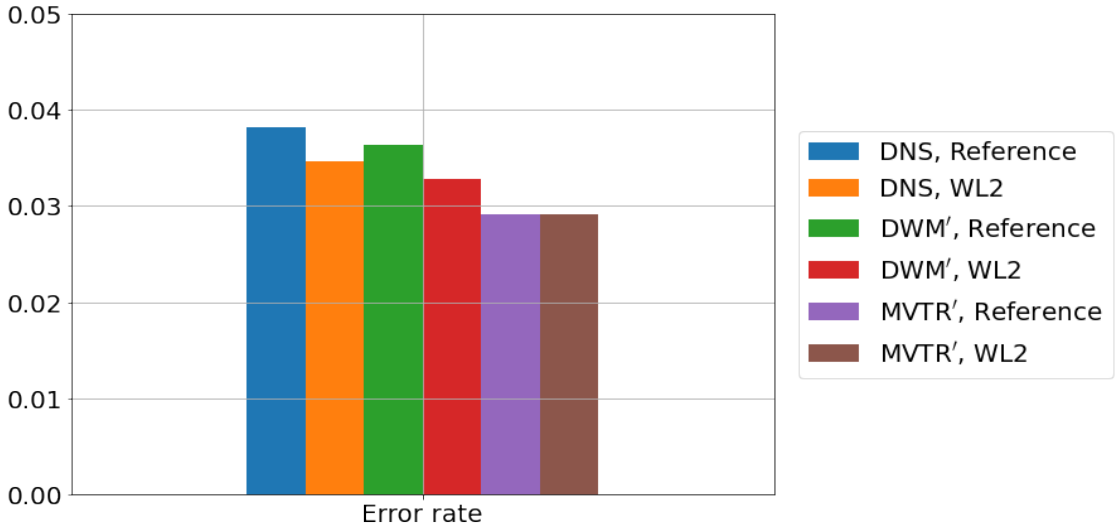
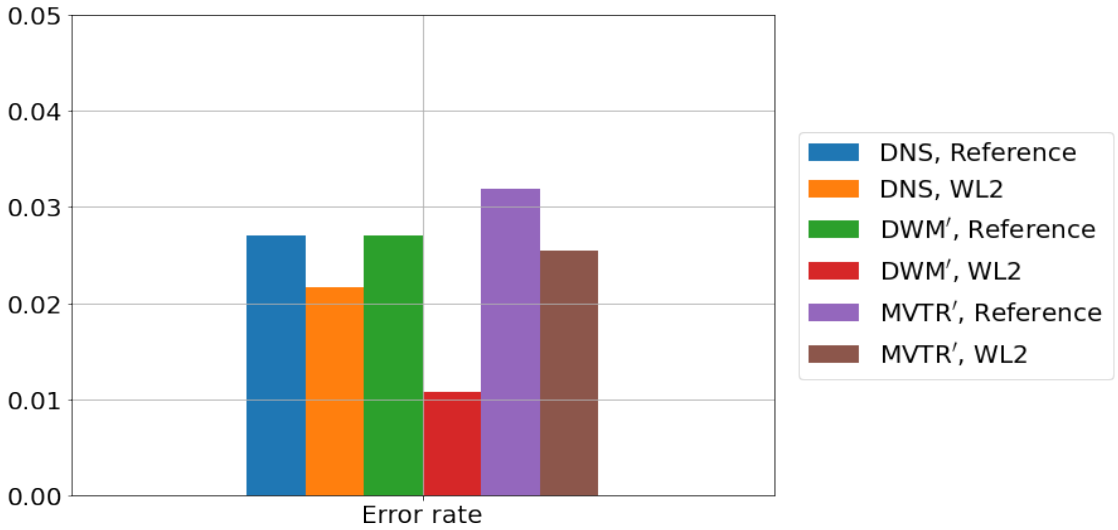


Figure 4.7: Performances of binary classification models using support vector machines with a radial basis functions (RBF svm). The metrics are shown with all test data, see Section 3.3. The 2-neighborhood graphs considered were either generated with Dark Web Market (DWM′) references enriched with DNS data, with Malware/Vulnerability Technical Reporting (MVTR′) references enriched with DNS data or just with DNS data (DNS), see Section 3.1.5. The features were generated by a vertex kernel in the Weisfeiler Lehman framework with two relabeling iterations and transformed by tf-idf transformers (WL2). For the reference models the features were generated with an edge kernel without Weisfeiler Lehman relabeling iterations and transformed by tf-idf transformers.



(a) Error rates on all test data put through the Weisfeiler Lehman graph kernel framework with a vertex kernel and 2 relabeling iterations (WL2), the reference models were implemented with an edge kernel without relabeling iterations. There were in total 549 seed vertices where 84 were malicious.



(b) Error rates on the strict test data put through the Weisfeiler Lehman graph kernel framework with a vertex kernel and 2 relabeling iterations (WL2), the reference models were implemented with an edge kernel without relabeling iterations. There were in total 157-185 seed vertices where 2-5 were malicious, depending on edge set.

Figure 4.8: Error rates of binary classification models using a RBF svm. The metrics are shown with all test data in Subfigure 4.8a and with the strict test data in Subfigure 4.8b, see Section 3.3. Since there were so few malicious seed vertices in the strict test set F_1 scores are not included.

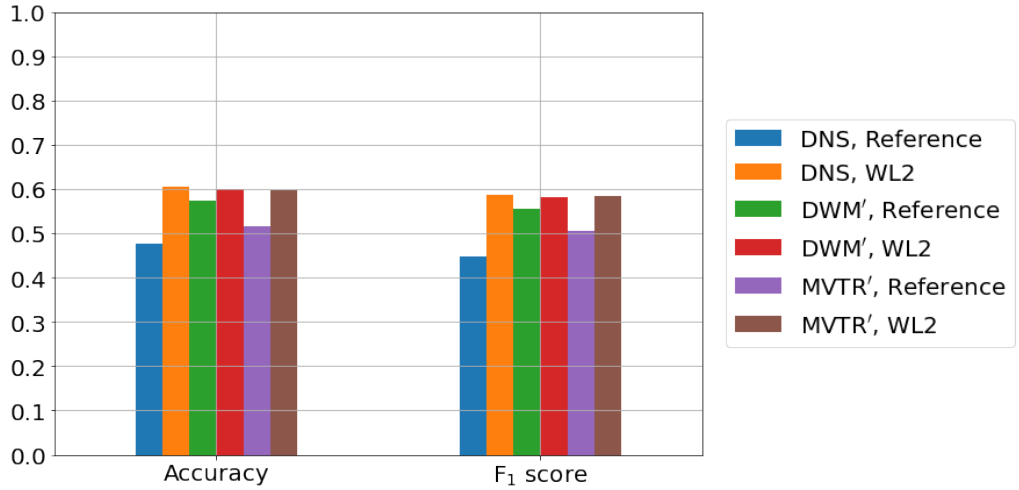
4.2.2 Multiclass classification problem

The results for the multiclass classification models are shown in Figure 4.9. The same model choices were made as for the binary case. That is WL2 means a vertex kernel in Weisfeiler Lehman framework with 2 relabeling iterations and the reference models were implemented with just an edge kernel. Similarly as for the binary model, Subfigure 4.9a displays the result on the whole test set, while Subfigure 4.9b displays the results on the strict test set. For a more extensive display of results, see Appendix A.2.

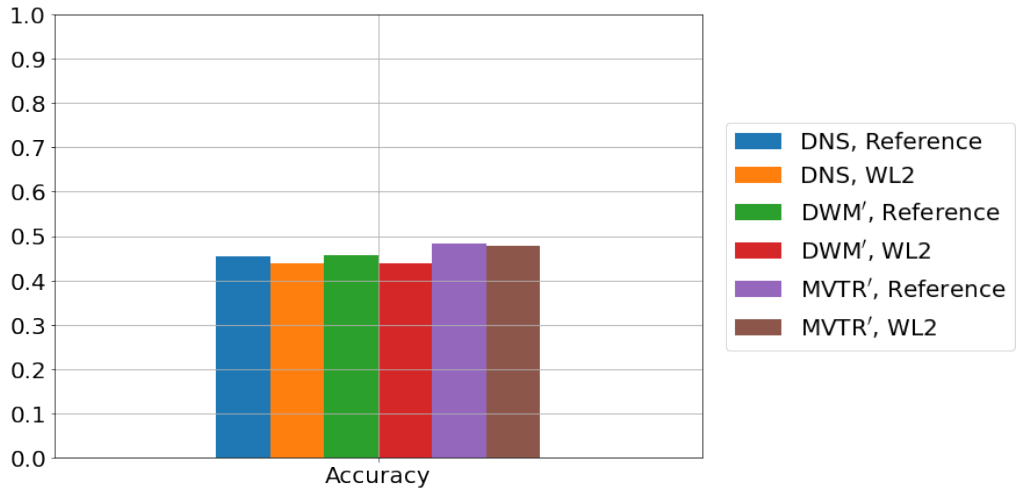
Noticeable is that the performance of the multiclass classification models was quite poor while for the binary classification models the performance was quite high. For one of these classification models the confusion matrix is shown in Table 4.3. Although it is visible also here that the performance was quite low, the ordinal aspect of the model seems to have worked better. That is, most misclassifications were to an adjacent risk score interval.

		Correct		
	Risk score interval	[0,25)	[25,65)	[65,100)
Estimated	[0,25)	69	72	50
	[25,65)	22	50	18
	[65,100)	65	69	134

Table 4.3: Confusion matrix for a multiclass logistic classification model performance on all test data. The 2-neighborhood graphs considered were either generated only with DNS data, hence the edge set was E^{DNS}. The features were generated by a vertex kernel in the Weisfeiler Lehman framework with two relabeling iterations and transformed by a tf-idf transformer.



(a) Accuracy and F_1 score on all test data put through the Weisfeiler Lehman graph kernel framework with a vertex kernel and 2 relabeling iterations (WL2), the reference models were implemented with an edge kernel without relabeling iterations.



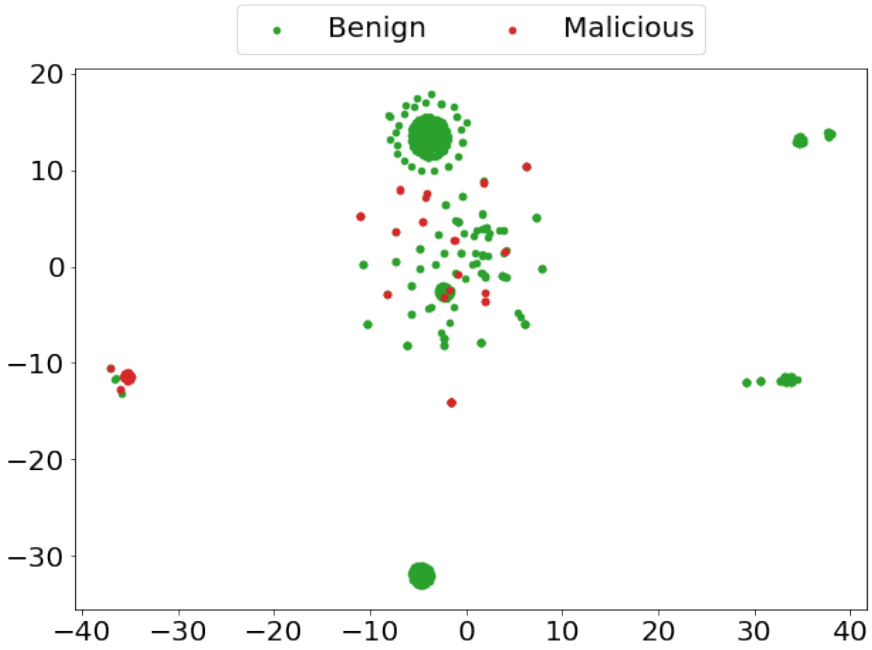
(b) Accuracy on the strict test data put through the Weisfeiler Lehman graph kernel framework with a vertex kernel and 2 relabeling iterations (WL2), the reference models were implemented with an edge kernel without relabeling iterations. Since there were so few samples belonging to the risk score interval $[25, 65)$, 0-18 depending on edge set, F_1 scores are omitted.

Figure 4.9: Performances of multiclass classification models using multiclass logistic regression. The 2-neighborhood graphs considered were either generated with Dark Web Market (DWM′) references enriched with DNS data, with Malware/Vulnerability Technical Reporting (MVTR′) references enriched with DNS data or just with DNS data (DNS), see Section 3.1.5. The features were generated by a vertex kernel in the Weisfeiler Lehman framework with two relabeling iterations and transformed by tf-idf transformers (WL2). For the reference models the features were generated with an edge kernel without Weisfeiler Lehman relabeling iterations and transformed by tf-idf transformers.

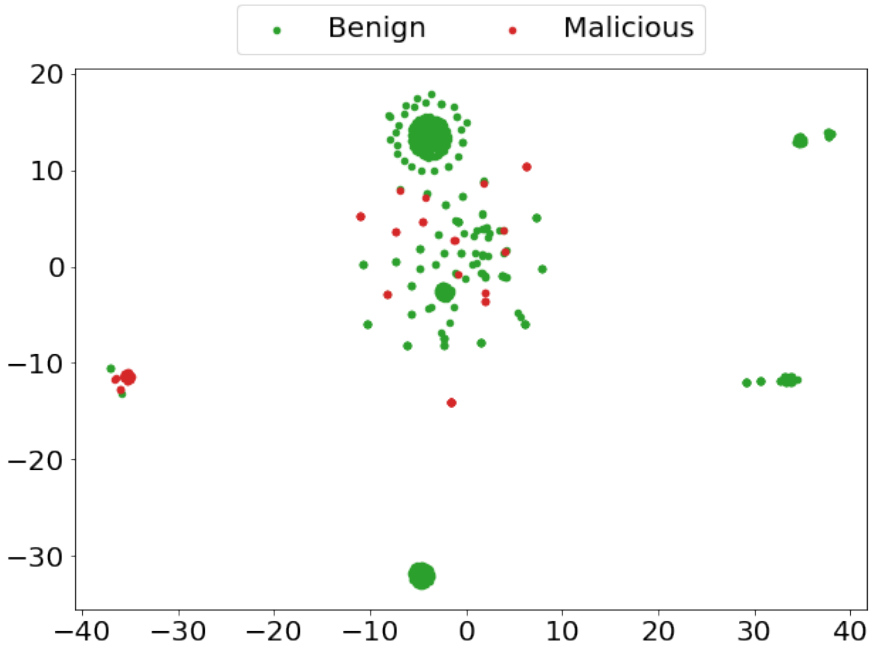
4.2.3 t-SNE

A clustering was performed on the features in order to display their structure. For the binary problem only features which corresponded to nonzero weights in a logistic regression model, see Table A.1, were used in the t-SNE projection [20]. This was done since very high dimensional data can cause issues with t-SNE, the methodology used here was implemented with inspiration from Shekhar et al [21]. The result of the projection on the test data is displayed in Figure 4.10. In Subfigure 4.10a the true labels were used for plotting while in Subfigure 4.10b the estimated labels were used. Note that these labels were not accessible for the clustering algorithm, they were only used for plotting.

For the multiclass problem all features were used, this was done because the accuracy was quite low for several models and hence the coefficients in the logistic model were not considered to be that informative. The t-SNE plots are displayed in Figure 4.11, with the same structure as in Figure 4.10. Comparing these figures it is clear that the multiclass problem was considerably harder, for the multiclass problem the true labels did not seem to group in clusters in Subfigure 4.11a while in Subfigure 4.10a they did.

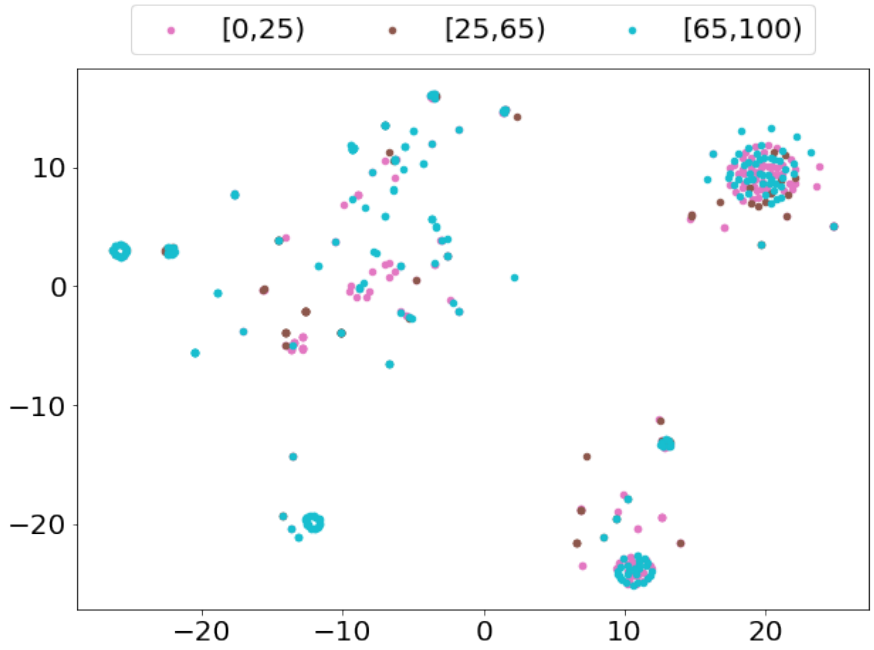


(a) True labels.

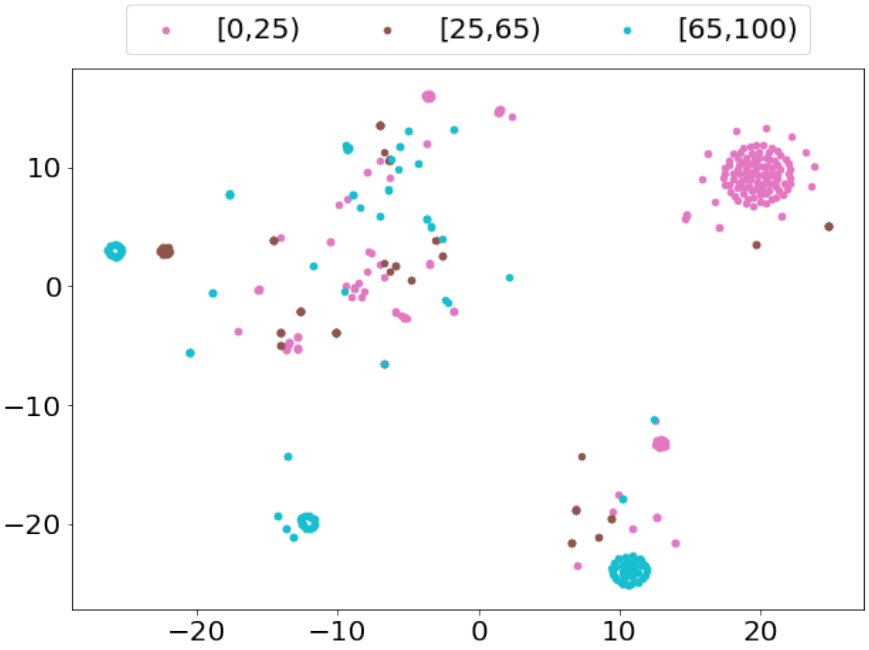


(b) Estimated labels with a RBF svm model.

Figure 4.10: t-SNE plots of the test 2-neighborhood graphs, with edge set $E^{MVTR'}$, put through two Weisfeiler Lehman iterations with a vertex kernel. Moreover only the features which corresponded to non-zero weight values in the logistic regression model were considered, see Appendix A.1



(a) True labels.



(b) Estimated labels with a multiclass logistic regression model.

Figure 4.11: t-SNE plots of the test 2-neighborhood graphs, with edge set E^{DNS}, put through two Weisfeiler Lehman iterations with a vertex kernel. Here all features were considered in the t-SNE projection.

5
Discussion

Since the entities used in this work already have risk scores, one might ask why the modeling approaches in this thesis were necessary. Firstly, it has been shown that using graph interpretations can yield accurate risk evaluations. Secondly, these modeling approaches could be used for new, or inaccurately evaluated, Internet Domains. Lastly, it has been shown that the intuitive graph context for entities on the Security Intelligence Graph (SIG) can be used in an algorithmic approach.

For the binary classification problem the reference models produced similar results as the models using the Weisfeiler Lehman framework, although generally somewhat worse considering F_1 score see Figure 4.7. Looking at the error rate for the strict test set, see Subfigure 4.8b, we see an improvement for the Weisfeiler Lehman models, suggesting that these models captured some important more complex structures than the more naive reference models.

5.1 Comparisons with other work

Comparing the results to other work is difficult. For Wang et al [3], Shin et al [4] and Carter et al [5] the output of their models are completely different. While for Xu et al [8] they mainly focus on the embedding they created and made comparisons with regard to similarity measures between different graphs. This was not our aim and hence a comparison is hard to make. Although their results are impressing and a similar approach could be applied in our classification setting as well. Tavabi et al [7] method can be considered most similar to our approach. They obtained a F_1 score of 0.80 for their prediction model, predicting if cyber vulnerabilities will be exploited. It is however not a fair comparison to compare our F_1 score of around 0.89 to theirs. Since their problem is quite different and can be considered harder to solve. Lastly, Jacobs et al [6] also considered predictions of if cyber vulnerabilities will be exploited. They presented their results with a Receiver Operator Characteristic (ROC) curve and a Precision/Recall curve along with values of the areas under respective curve, making it difficult to compare with our metrics. Moreover, they are solving a potentially harder problem.

5.2 Test set

As discussed in Section 3.3, two test sets were used to evaluate the models. Firstly, the test set X_{test} which consisted of 20 % randomly selected graphs, which were

present in the data set. Secondly, the strict test set $X_{test}^{strict} \subset X_{test}$. Where seed vertices in the training set were not allowed in the 2-neighborhood graphs, see Figure 3.4. This restriction lead to a considerably smaller strict test set.

The main issue with X_{test}^{strict} is that it is too small. The restriction made reduces the test set with about 2/3, more importantly for the binary problem it only leaves 5 malicious seed vertices or less. Hence all metrics regarding only positives, that is recall, precision and F_1 score contain a lot of variance and are not informative. Regarding accuracy, we can be fairly certain that the model did not generate a lot of false positives, regarding false negatives nothing can be concluded even though the results suggest that false negatives seems to be an issue. See Table A.3 for results on the strict test set.

An issue with X_{test} is that it contains information used in the training phase for the models. The connectivity of the data is however an inherent property of graph data, at least when considering neighborhood graphs. It is also likely that this will occur in a real scenario. In Figure 3.4, the image was generated by querying the Neo4j database for two seed vertices appearing in each others 2-neighborhoods. Here a third seed vertex appeared by chance, illustrating that when considering graph data it is quite likely that vertices of interest will appear in each others neighborhoods. Moreover caution has been taken in order to not influence the classification, seed vertices which appear in the training data were given an benign label independently of its true label. The decision to set the label to benign, or the lowest label in the multiclass setting, was motivated by the fact that the model should underestimate rather than overestimate risk. Another possible solution would be to explicitly label these vertices with a separate label, for example with label *evaluating*. This approach was however discarded as this label would not appear in a real-world scenario. At least not when evaluating 2-neighborhood graphs one by one.

5.3 Labels

Throughout the modeling the labels, that is entities' risk scores, have been consider as holding ground truth. Even though it is known that some entities with lower risk score should have higher risk scores. In a production implementation it is mainly identifying these entities which are of interest. Hence the evaluation done of the models might not actually capture the intended purpose of the models. Especially high precision might not be ideal. High accuracy following this reasoning would not be the best possible output but rather a ranking of Internet Domains, with the most urgent to evaluate highest in the ranking, would be an ideal output. However these kinds of considerations would demand a manual evaluation of the entities considered, these evaluations are beyond the scope of the thesis. Although they should be considered in order to make a final evaluation of a model before implementing it in a real-world scenario.

In the binary classification setting the results, see Figure 4.7, are promising. However it might be the case that Internet Domains which have high risk scores (≥ 80)

are quite simple to identify. Hence these promising results should be interpreted with some caution.

The multiclass classification methods performances were quite a lot worse than in the binary setting, see Figure 4.9. That the metrics should be lesser than for the binary setting is expected since the problem is more difficult. However the large difference is something that should be investigated further, it might be the case that the modeling approach simply does not well capture the structure of the problem. Something that Figure 4.11 suggests as the true labels do not seem to cluster well. Another possible reason would be that we were actually learning something about the context and that there were issues with the initial labeling. These issues should be evaluated further, if the multiclass version of the labeling problem should be implemented.

5.4 Time aspect

The time aspect of the data set was not explicitly included. It was implicitly considered in the sense that only references within the time period 2020-08-01 to 2020-11-17 were included in the data set. Some of the risk rules have a 60 days validity, that is in order to trigger them the triggering event must have occurred within 60 days of the current risk evaluation. Hence the time span, 109 days, can be considered as too large. The reason for choosing a larger rather than smaller time span was that initially, without including DNS data as well, the reference data did not give rise to meaningful 2-neighborhood graphs, explicitly 2-neighborhood graphs only including the seed vertex. Hence the choice of including more rather than less data seemed sensible. It was unfortunately also quite time consuming to set up the database and generate the graph structure, hence experimenting with different time periods was omitted from the modeling approaches.

The models were not meant to be predicting labels but rather evaluating the context entities are currently in on the SIG and see if such contexts can yield good risk evaluations. Hence issues with time traveling are not as prevalent as for a predictive models. However having an evaluation time for the entities would further aid in handling this potential problem. This was not accessible in the data set considered in our work.

5.5 Vertex and edge features

When trying different modeling approaches vertex features for seed vertices were considered. Many of these features trigger risk rules. Hence including them could lead to model which just mimics risk rules and this would result in a nonsense model. However, some of the features counts mentions of entities on different forms, such as dark web or social media. These were not triggering risk rules and could have benefited the model. Although, when using only these features in a support vector

machine the performance was very bad for Internet Domains on the validation set. Hence these features were not included in the final models.

Vertex features can not be included in the Weisfeiler Lehman framework, although there are graph kernels which can include them. One such example is the GraphHopper kernel, introduce by Feragen et al [22]. However, when using the implementation in GraKel [23], processing the graphs considered took over a week, even when excluding the very large graphs. Hence it was not feasible to use this approach with the computational resources accessible during our work. This resulted in that more complex graph kernels which consider vertex features were not used. Another reason to not use more computationally demanding kernels is that such models would probably never be implemented in a real application.

Using the edge kernel, edge weights can be considered, see (2.18). This was implemented with the weight function w corresponding to a count of number of references which connected two entities. Implementing such a counter added complexity mainly in the data collection step, see Section 3.1.4. By considering results on the validation set, this added complexity did not seem to improve performance. Hence edge weights were not considered in the final models.

5.6 Complexity

One advantage with the Weisfeiler Lehman framework is that the relabeling procedure only has runtime complexity in $\mathcal{O}(n^2)$, where n is the maximal number of vertices in the graphs considered [14]. Hence implementing these models would be feasible in a real world application.

The runtime in $\mathcal{O}(n^2)$ is however without regarding the complexity for generating the 2-neighborhood graphs. Using a graph database such as neo4j makes these queries fast and straightforward to write. The Security Intelligence Graph (SIG) is however not implemented in a graph database and keeping a copy of it in a graph database does not seem like a feasible idea, because of the size of the database. Transforming the whole database into a graph database would be a possible approach, although there are several disadvantages with this, the most apparent one is that the database should be optimized for other, non-graph based, queries as well. Another reason is that it is very costly and demands a lot of work migrating a database of this size.

In order to implement the proposed modeling, the 2-neighborhood graph would have to be generated straight from the SIG. As this corresponds to the data generating steps in Figure 3.3, this yields quite complex and demanding queries.

By considering Figure 4.7, one can see that only considering DNS data gave similar results as including Malware/Vulnerability Technical Reporting data or Dark Web Market data. Although the performances are slightly worse, these results motivates an implementation using only DNS data. Only considering DNS data would make the implementation easier, since then the whole procedure of doing references

queries, as in Figure 3.3, could be omitted.

5.7 Implementation suggestions

If an approach similar to the one made in thesis should be implemented for entities on SIG, only DNS data should be used for generating edges. The SIG would be used for getting risk scores. This, the results suggest, would reduce query complexity without significantly reducing performance. Lastly the Weisfeiler-Lehman framework with a vertex kernel should be used, since the vertex kernel performs similarly as the edge kernel while being more memory efficient, see Section 2.6.2. An alternative would be to only use an edge kernel, only using a vertex kernel reduces performance see Table A.2. This could lead to a larger memory demand but reduces computational complexity. However the results marginally suggest that the Weisfeiler-Lehman relabeling iterations capture some important structures for classification, which are not captured when only use an edge kernel, see Subfigure 4.8a.

5.8 Future work

An approach similar to only using DNS data for internet Domains could be done for IP Addresses using who-is data. Who-is information can be accessed by querying a who-is service with an IP address which yields a response corresponding to a company or a person. There are publicly available such services, for example RIPE [24]. This process could also be done with a reverse DNS lookup setup.

The ontology part of the SIG has not been considered in our work. This choice was made in order to decrease complexity in the data collection part. However the ontology part of the SIG holds a lot of important information, to include this information might improve the classification and should be investigated further.

A deep learning approach for embedding the 2-neighborhood graphs in a vector space, similar to the one made by Tavabi et al [7], can be used to evaluate the context for entities on SIG. This approach has the advantage that an application's run time would have linear complexity rather than quadratic [7].

Including the time when a vertex is introduced to the data set as well as having time information regarding when edges were generated could lead to more complex modeling. Something which might improve the models. This kind of time data is available in the SIG. Considering this data is something which could be investigated further.

6
Conclusion

In this work, we have derived a methodology for classification of Internet Domains based on their neighborhood features in the Security Intelligence Graph (SIG). The results suggest that using these features yields good classification for a binary classification problem. Furthermore, the results suggest that it is possible to perform good classification only using Domain Name Server information around Internet Domains, along with the risk scores provided in the SIG. A more fine-grained estimation did not produce equally good results. The results for the binary classification models suggest that using an edge kernel with a radial basis function support vector machine yields similar results as using a Weisfeiler-Lehman graph kernel. On the contrary, for the more fine-grained estimation models, or multiclass models, the Weisfeiler-Lehman graph kernel framework seemed superior to the simpler edge kernels. Although the results with Weisfeiler-Lehman graph kernels for the multiclass problem were considerably worse compared to the binary classification problem.

Bibliography

[1] Lyrvall, B. (FRA), Hallin, L. (The Swedish Armed Forces), Eliasson, D. (MSB), & Friberg, K. (SÄPO) (2020). Debattartikel: Beslut om inrättande av nationellt center för sybersäkerhet - Säkerhetspolisen. https://www.sakerhetspolisen.se/ovrigt/pressrum/aktuellt/aktuellt/2020-12-10-beslut-om-inrattande-av-nationellt-center-for-cybersakerhet/debattartikel-beslut-om-inrattande-av-nationellt-center-for-sybersakerhet.html

[2] Talesh, S. (2018). Data Breach, Privacy, and Cyber Insurance: How Insurance Companies Act as "Compliance Managers" for Businesses. Law & Social Inquiry, 43(2), 417-440. doi:10.1111/lsi.12303

[3] Wang, J., Neil, M., & Fenton, N. (2020). A Bayesian network approach for cybersecurity risk assessment implementing and extending the FAIR model. Computers & Security, 89, 101659. https://doi.org/https://doi.org/10.1016/j.cose.2019.101659

[4] Shin, J., Son, H., Khalil Ur, R., & Heo, G. (2015). Development of a cyber security risk model using Bayesian networks. Reliability Engineering and System Safety, 134, 208–217. https://doi.org/10.1016/j.ress.2014.10.006

[5] K. M. Carter, N. Idika and W. W. Streilein. Probabilistic Threat Propagation for Network Security. IEEE Transactions on Information Forensics and Security, vol. 9, no. 9, pp. 1394-1405, Sept. 2014. doi:10.1109/TIFS.2014.2334272.

[6] Jacobs, J., Romanosky, S., Edwards, B., Roytman, M., & Adjerid, I. (2019). Exploit Prediction Scoring System (EPSS). BlackHat 2019. http://arxiv.org/abs/1908.04856

[7] Tavabi, N., Goyal, P., Almukaynizi, M., Shakarian, P., & Lerman, K. (2018). Darkembed: Exploit prediction with neural language models. 32nd AAAI Conference on Artificial Intelligence, AAAI 2018, 7849–7854. https://usc-isi-i2.github.io/papers/tavabi18-aaai.pdf

[8] Xu, X., Liu, C., Feng, Q., Yin, H., Song, L., & Song, D. (2017). Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection. Proceedings of the ACM Conference on Computer and Communications Security, 363–376. https://doi.org/10.1145/3133956.3134018

[9] The Security Intelligence Graph: Inside Recorded Future's Methodology and Patented Technology (2019). Accessed 20-01-2021: https://www.recordedfuture.com/resources/reports/

[10] Bishop, C. M. (2006). Pattern Recognition and Machine Learning (M. Jordan, J. Kleinberg, & B. Schölkopf (eds.)). Springer.

https://www.microsoft.com/en-us/research/uploads/prod/2006/01/Bishop-Pattern-Recognition-and-Machine-Learning-2006.pdf

[11] M. Mohri, A. Rostamizadeh, and A. Talwalkar. Foundations of Machine Learning. MIT Press, 2012.

[12] Andréasson, N., Evgrafov, A. and Patriksson, M. (2016) An introduction to continuous optimization: foundations and fundamental algorithms, page 172. Third edition adjusted. Studentlitteratur.

[13] Kriege, N. M., Johansson, F. D., & Morris, C. (2019). A Survey on Graph Kernels. Applied Network Science, 5(1). https://doi.org/10.1007/s41109-019-0195-3

[14] Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., & Borgwardt, K. M. (2011). Weisfeiler-Lehman Graph Kernels. In Journal of Machine Learning Research (Vol. 12). https://jmlr.org/papers/volume12/shervashidze11a/shervashidze11a.pdf

[15] Pedregosa, F., Michel, V., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Vanderplas, J., Cournapeau, D., Pedregosa, F., Varoquaux, G., Gramfort, A., Thirion, B., Grisel, O., Dubourg, V., Passos, A., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830. http://scikit-learn.sourceforge.net

[16] Carrillo, Henry & Brodersen, Kay H. & Castellanos, Jose. (2014). Probabilistic Performance Evaluation for Multiclass Classification Using the Posterior Balanced Accuracy. Advances in Intelligent Systems and Computing. 252. 347-361. 10.1007/978-3-319-03413-3_25.

[17] Rajaraman, A., & Ullman, J. D. (2011). Data Mining. Pages 1-17. https://doi.org/10.1017/CBO9781139058452.002

[18] Sokolova, M. & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. Information Processing & Management, Volume 45, Issue 4, Pages 427-437, ISSN 0306-4573. https://doi.org/10.1016/j.ipm.2009.03.002.

[19] Chakrabarti, D., & Faloutsos, C. (2012). Graph Mining: Laws, Tools, and Case Studies. In Synthesis Lectures on Data Mining and Knowledge Discovery (Vol. 3, Issue 3). https://doi.org/10.2200/s00449ed1v01y201209dmk006

[20] van der Maaten, L.J.P. & Hinton, G.E. (2008). Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research 9(Nov):2579-2605.

[21] Shekhar, K., Lapan, S. W., Whitney, I. E., Tran, N. M., Macosko, E. Z., Kowalczyk, M., Adiconis, X., Levin, J. Z., Nemesh, J., Goldman, M., McCarroll, S. A., Cepko, C. L., Regev, A., & Sanes, J. R. (2016). Comprehensive Classification of Retinal Bipolar Neurons by Single-Cell Transcriptomics. Cell, 166(5), 1308-1323.e30. https://doi.org/https://doi.org/10.1016/j.cell.2016.07.054

[22] Feragen, A., Kasenburg, N., Petersen, J., De Bruijne, M., & Borgwardt, K. (2013). Scalable kernels for graphs with continuous attributes. Advances in Neural Information Processing Systems.

[23] Siglidis, G., Giatsidis, C., & Skianis, K. (2020). GraKeL: A Graph Kernel Library in Python. In Journal of Machine Learning Research (Vol. 21, Issue 54). http://jmlr.org/papers/v21/18-370.html.
[24] RIPE: https://www.ripe.net/about-us. Accessed: 31-01-2020.

A

Appendix 1

A.1 The relabeling function in Example 2.6.4

Here the relabeling function from the Weisfeiler Lehman relabeling example 2.6.4 is outlined. The algorithm used is shown in Algorithm 1. The relabeled graphs can be found in Figures 2.2 and 2.3.

Relabel 0. Initialize l_{re} as an empty `hashmap` and set $counter = 0$.
Relabel 1. Consider labels $\{b, m\}$:
- I. b not in l_{re} :
  - i. Set $l_{re}(b) = counter$.
  - ii. Set $counter = counter + 1 = 1$.
  - iii. **return** $l_{re}(b) = 0$.
- II. m not in l_{re}:
  - i. Set $l_{re}(b) = counter$.
  - ii. Set $counter = counter + 1 = 2$.
  - iii. **return** $l_{re}(m) = 1$.

Relabel 2. Consider labels $\{"0, 001", "1, 00", "0, 00", "0, 00111", "1, 0"\}$:
- I. $"0, 001"$ not in l_{re}:
  - i. Set $l_{re}("0, 001") = counter$.
  - ii. Set $counter = counter + 1 = 3$.
  - iii. **return** $l_{re}("0, 001") = 2$.
- II. $"1, 00"$ not in l_{re}:
  - i. Set $l_{re}("1, 00") = counter$.
  - ii. Set $counter = counter + 1 = 4$.
  - iii. **return** $l_{re}("1, 00") = 3$.
- III. $"0, 00"$ not in l_{re}:
  - i. Set $l_{re}("0, 00") = counter$.
  - ii. Set $counter = counter + 1 = 5$.
  - iii. **return** $l_{re}("0, 00") = 4$.
- IV. $"0, 00111"$ not in l_{re}:
  - i. Set $l_{re}("0, 00111") = counter$.
  - ii. Set $counter = counter + 1 = 6$.
  - iii. **return** $l_{re}("0, 00111") = 5$.
- V. $"1, 0"$ not in l_{re}:
  - i. Set $l_{re}("1, 0") = counter$.
  - ii. Set $counter = counter + 1 = 7$.

iii. **return** $l_{re}(\text{"}1,0\text{"}) = 6$.

Relabel 3. Consider labels $\{\text{"}2,234\text{"}, \text{"}3,245\text{"}, \text{"}4,25\text{"}, \text{"}5,23466\text{"}, \text{"}6,5\text{"}\}$:

I. "$2,234$" not in l_{re}:
   i. Set $l_{re}(\text{"}2,234\text{"}) = counter$.
   ii. Set $counter = counter + 1 = 8$.
   iii. **return** $l_{re}(\text{"}2,234\text{"}) = 7$.

II. "$3,245$" not in l_{re}:
   i. Set $l_{re}(\text{"}3,245\text{"}) = counter$.
   ii. Set $counter = counter + 1 = 9$.
   iii. **return** $l_{re}(\text{"}3,245\text{"}) = 8$.

III. "$4,25$" not in l_{re}:
   i. Set $l_{re}(\text{"}4,25\text{"}) = counter$.
   ii. Set $counter = counter + 1 = 10$.
   iii. **return** $l_{re}(\text{"}4,25\text{"}) = 9$.

IV. "$5,23466$" not in l_{re}:
   i. Set $l_{re}(\text{"}5,23466\text{"}) = counter$.
   ii. Set $counter = counter + 1 = 11$.
   iii. **return** $l_{re}(\text{"}5,23466\text{"}) = 10$.

V. "$6,5$" not in l_{re}:
   i. Set $l_{re}(\text{"}6,5\text{"}) = counter$.
   ii. Set $counter = counter + 1 = 12$.
   iii. **return** $l_{re}(\text{"}6,5\text{"}) = 11$.

Relabel 4. Consider labels $\{b, m\}$:

I. b in l_{re} :
   i. **return** $l_{re}(b) = 0$.

II. m in l_{re}:
   i. **return** $l_{re}(m) = 1$.

Relabel 5. Consider labels $\{\text{"}0,001\text{"}, \text{"}1,000\text{"}, \text{"}0,00111\text{"}, \text{"}1,01\text{"}\}$:

I. "$0,001$" in l_{re} :
   i. **return** $l_{re}(\text{"}0,001\text{"}) = 2$.

II. "$1,000$" not in l_{re}:
   i. Set $l_{re}(\text{"}1,000\text{"}) = counter$.
   ii. Set $counter = counter + 1 = 13$.
   iii. **return** $l_{re}(\text{"}1,000\text{"}) = 12$

III. "$0,00111$" in l_{re} :
   i. **return** $l_{re}(\text{"}0,00111\text{"}) = 5$

IV. "$1,01$" not in l_{re}:
   i. Set $l_{re}(\text{"}1,01\text{"}) = counter$.
   ii. Set $counter = counter + 1 = 14$.
   iii. **return** $l_{re}(\text{"}1,01\text{"}) = 13$

Relabel 6. Consider labels $\{\text{"}2,2512\text{"}, \text{"}12,225\text{"}, \text{"}5,22121313\text{"}, \text{"}13,513\text{"}\}$:

I. "$2,2512$" not in l_{re}:
   i. Set $l_{re}(\text{"}2,2512\text{"}) = counter$.
   ii. Set $counter = counter + 1 = 15$.
   iii. **return** $l_{re}(\text{"}2,2512\text{"}) = 14$.

II. "$12,225$" not in l_{re}:

i. Set $l_{re}(\text{"}12, 225\text{"}) = counter$.
ii. Set $counter = counter + 1 = 16$.
iii. **return** $l_{re}(\text{"}12, 225\text{"}) = 15$.

III. "$5, 22121313$" not in l_{re}:
i. Set $l_{re}(\text{"}5, 22121313\text{"}) = counter$.
ii. Set $counter = counter + 1 = 17$.
iii. **return** $l_{re}(\text{"}5, 22121313\text{"}) = 16$.

IV. "$13, 513$" not in l_{re}:
i. Set $l_{re}(\text{"}13, 513\text{"}) = counter$.
ii. Set $counter = counter + 1 = 18$.
iii. **return** $l_{re}(\text{"}13, 513\text{"}) = 17$.

A.2 Results

The abbreviations used here are defined in Sections 4.1.1 and 4.2.1. One thing we notice when comparing Table A.1 and Table A.2 is that it did not appear to be possible to find a good linearly separating hyperplane without Weisfeiler-Lehman iterations. Combining a support vector machine with a radial basis function (RBF svm) did on the contrary seem to work well without Weisfeiler-Lehman iterations. We also notice that with two Weisfeiler-Lehman iterations the logistic model and the RBF svm performed very similarly, suggesting that here it was possible to find a linearly separating hyperplane.

Data and model	Kernel	Accuracy	Precision	Recall	F_1 score
DNS, Reference	edge	0.81	0.45	0.89	0.60
DNS, Reference	vertex	0.92	0.68	0.89	0.77
DNS, WL2	edge	0.96	0.83	0.93	0.88
DNS, WL2	vertex	0.96	0.84	0.94	0.89
DWM′, Reference	edge	0.81	0.45	0.89	0.60
DWM′, Reference	vertex	0.92	0.68	0.89	0.77
DWM′, WL2	edge	0.96	0.84	0.93	0.88
DWM′, WL2	vertex	0.97	0.88	0.94	0.91
MVTR′, Reference	edge	0.59	0.19	0.54	0.29
MVTR′, Reference	vertex	0.89	0.58	0.89	0.70
MVTR′, WL2	edge	0.97	0.86	0.94	0.90
MVTR′, WL2	vertex	0.97	0.90	0.94	0.92

Table A.1: Performances of binary models on the whole test set implemented with a logistic regression model.

Data and model	Kernel	Accuracy	Precision	Recall	F_1 score
DNS, Reference	edge	0.96	0.85	0.90	0.88
DNS, Reference	vertex	0.94	0.77	0.90	0.83
DNS, WL2	edge	0.97	0.87	0.90	0.89
DNS, WL2	vertex	0.97	0.87	0.90	0.89
DWM′, Reference	edge	0.96	0.86	0.90	0.88
DWM′, Reference	vertex	0.95	0.78	0.90	0.84
DWM′, WL2	edge	0.97	0.88	0.90	0.89
DWM′, WL2	vertex	0.97	0.87	0.93	0.90
MVTR′, Reference	edge	0.97	0.90	0.90	0.90
MVTR′, Reference	vertex	0.95	0.81	0.90	0.85
MVTR′, WL2	edge	0.97	0.90	0.90	0.90
MVTR′, WL2	vertex	0.97	0.90	0.90	0.90

Table A.2: Performances of binary models on the whole test set implemented with a radial basis function support vector machine.

Data and model	Kernel	Accuracy	Precision	Recall	F_1 score
DNS, Reference	edge	0.97	0.50	0.20	0.29
DNS, Reference	vertex	0.97	0.33	0.20	0.25
DNS, WL2	edge	0.98	1.00	0.20	0.33
DNS, WL2	vertex	0.98	1.00	0.20	0.33
DWM′, Reference	edge	0.97	0.50	0.20	0.29
DWM′, Reference	vertex	0.97	0.33	0.33	0.33
DWM′, WL2	edge	0.98	1.00	0.20	0.33
DWM′, WL2	vertex	0.99	1.00	0.60	0.75
MVTR′, Reference	edge	0.97	0.50	0.20	0.29
MVTR′, Reference	vertex	0.95	0.20	0.20	0.20
MVTR′, WL2	edge	0.97	1.00	0.20	0.33
MVTR′, WL2	vertex	0.97	1.00	0.20	0.33

Table A.3: Performances of binary models on the strict test set implemented with a radial basis function support vector machine. There were in total 157-185 seed vertices where 2-5 were malicious, depending on edge set. Hence precision, recall and F_1 score should be interpreted with caution. Although the models does not seem to correctly identify these few malicious seed vertices.

For the multiclass problem there are clear improvements using a Weisfeiler-Lehman graph kernel compared to the simpler graph kernels, see Table A.4. On the strict test set the performances are considerably worse for both the reference models and the models with 2 iterations in the Weisfeiler-Lehman graph kernel framework, see Table A.5.

Data and model	Kernel	Accuracy	Precision	Recall	F_1 score
DNS, Reference	edge	0.48	0.46	0.44	0.45
DNS, Reference	vertex	0.46	0.46	0.46	0.46
DNS, WL2	edge	0.58	0.58	0.56	0.57
DNS, WL2	vertex	0.60	0.60	0.58	0.59
DWM′, Reference	edge	0.57	0.58	0.53	0.56
DWM′, Reference	vertex	0.53	0.54	0.51	0.52
DWM′, WL2	edge	0.59	0.58	0.57	0.58
DWM′, WL2	vertex	0.60	0.59	0.57	0.58
MVTR′, Reference	edge	0.52	0.51	0.50	0.51
MVTR′, Reference	vertex	0.46	0.48	0.49	0.49
MVTR′, WL2	edge	0.58	0.59	0.59	0.59
MVTR′, WL2	vertex	0.60	0.58	0.59	0.58

Table A.4: Performances of multiclass models on the whole test set implemented with a multiclass logistic regression model.

Data and model	Kernel	Accuracy	Precision	Recall	F_1 score
DNS, Reference	edge	0.45	0.41	0.35	0.38
DNS, Reference	vertex	0.43	0.37	0.34	0.36
DNS, WL2	edge	0.44	0.47	0.34	0.39
DNS, WL2	vertex	0.44	0.44	0.34	0.38
DWM′, Reference	edge	0.46	0.72	0.35	0.47
DWM′, Reference	vertex	0.48	0.35	0.34	0.35
DWM′, WL2	edge	0.44	0.47	0.34	0.39
DWM′, WL2	vertex	0.44	0.44	0.34	0.38
MVTR′, Reference	edge	0.48	0.33	0.34	0.34
MVTR′, Reference	vertex	0.46	0.38	0.32	0.35
MVTR′, WL2	edge	0.48	0.34	0.33	0.34
MVTR′, WL2	vertex	0.48	0.36	0.33	0.35

Table A.5: Performances of multiclass models on the strict test set implemented with a multiclass logistic regression model.