Masaryk University
Faculty of Informatics

The Categorization of the Darkweb

Master's Thesis

Bc. Linda Hansliková

Brno, 2020

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Linda Hansliková

Advisor: RNDr. Martin Stehlík, Ph.D.

Acknowledgement

My thanks go to my adviser RNDr. Martin Stehlík, Ph.D. for allowing me to proceed with this topic, for his guidance, invaluable advice, and patience. I thank Matej Pavla for his emotional support and for motivating me. I am also thankful to Jozef Vilkolák for the mathematical advice he has given me.

Abstract

The ordinarily accessible online world is composed of commonly legal, interlinked sites and pages. The structure of the dark web, however, may be different because of the nature of this hidden part of the Internet. Questions about the interconnection and the categorical makeup of the dark web content arise.

We were interested in answering these questions for a portion of the dark web. The results of our work include a classification model which categorizes the dark web pages with an accuracy of 88.2%. We compared two community detection algorithms and determined the Leiden algorithm to be more suited for the fulfilling of our goal. We visualized the pages divided into communities in form of a web graph.

The final web application is to our knowledge, at the time of writing this thesis, the only publicly available software to categorize the dark web content, display a comprehensible and zoomable web graph with information about category makeup, and the option of downloading further page details.

Keywords

dark web, artificial neural networks, supervised learning, community detection, web graph, web application

Contents

List of Figures

1 Introduction

The Internet is composed of the clear web, which contains indexed content and is accessible via standard web browsers. However, it is also composed of the dark web. Content in the dark web is not indexed nor accessible through special browsers. The priority of the dark web is anonymity.

Anonymity and security is meaningful to whistle-blowers as their actions may have severe consequences otherwise. Human-rights activists also risk persecution in oppressed parts of the world and therefore benefit from keeping their identity secret. Nevertheless, with anonymity comes a lack of accountability. Accountability is, however, important for the prevention of illegal activities. The dark web is thus also an environment with illegal content and services, e.g. child pornography, identity theft, or money laundering [8]. The monitoring of the dark web is helpful in shutting down such services.

An industrial partner of the Masaryk University was interested in the structure of the dark web. More specifically, information about the structure, page categories and content was sought. An application able to supply the partner with the before mentioned information needed to be developed. Additionally, the partner expected the application to run on UNIX systems. This thesis was aimed to provide the partner with the above mentioned application.

A data set was acquired by Juraj Noge as part of his bachelor's thesis [65]. The data set comprised pages scraped from the dark web and was stored in Elasticsearch. These pages were made available to us and would constitute the data set used in our application.

In this work we describe the dark web and analyse the provided data set in Chapter 2. The quantity of the pages represented several obstacles. The difficulty related to the classification is explained in Chapter 3. Machine learning and the approach chosen for the categorization of pages is also discussed in the chapter. Another complication, related to the displaying of the data is mentioned in Chapter 4. In addition, the chapter outlines what a web graph is and introduce two algorithms for community detection, the Louvain and the Leiden algorithm. Chapter 5 details how the gathered knowledge in the pre-

vious chapters is applied in the implementation of the application. We compare the output of the two community detection algorithms in Chapter 6. This chapter also compared the accuracy achieved by the classification model trained on different learning data sets. The last Chapter 7 states the results of this work and suggests future improvements.

2 Data set analysis

The data set we were working with was created by Juraj Noge as part of his bachelor's thesis [65]. The data consisted of pages which had been scraped[1] from the dark web utilising Ahmia [69] and stored in Elasticsearch (ES) [13] [65].

This chapter describes the clear web, deep web, and the dark web. The structure of the stored pages is listed, Ahmia and Elasticsearch are intriduced.

The Internet comprises networks from all over the world. The networks are connected and together they compose a global network. The Internet can be divided into the clear web, characterized in Section 2.1, and the deep web, introduced in Section 2.2 [9].

2.1 Clear web

The content of the clear web is indexed by search engines and is publicly accessible. The users are identified by their IP addresses and are usually not anonymous unless some privacy tools are used. The most used browsers worldwide according to market share are Chrome, Safari and Firefox [74]. The size of the clear web is difficult to determine. However, it is estimated to contain about 5.93 billion pages as of March 2020 [22].

2.2 Deep web

The content of the deep web is not indexed by search engines and is not accessible publicly. Emails or medical information are examples of resources in the deep web. A user needs to be authorized to access this data. It is more problematic to measure the size of the deep web because the content is not indexed. There are not many sources de-

1. Web scraping is a method leveraged for the collecting of data from web pages with automated software rather than doing so manually. The software used for scraping loads a website, stores the desired data along with links to other pages and then loads those pages.

picting the size of the deep web. In one article from 2012 the size was estimated to be 4,000 to 5,000 times larger than the clear web [19].

A portion of the deep web is the *dark web*, also called darknet.

2.2.1 Dark web

The dark web was designed to provide secure anonymity to users. It is therefore utilised by human-rights activists or whistle-blowers for accessing or publishing censored material, e.g. the Bible[2]. It is also leveraged for disclosing illegal material, for example child pornography. An additional usage is the trade with illegal goods, such as drugs or guns. Another way to exploit the dark web is to offer or order illegal services, for instance money laundering, hacking or murder [8]. The Onion Router (Tor) or the Invisible Internet Project (I2P) are two of the networks composing the darknet. These networks are described more closely in Subsection 2.2.2 and Subsection 2.2.3 respectively.

2.2.2 Tor

The Tor network [71] is accessible through the Tor browser or Tor proxy. Tor makes use of *onion routing* (ORing) [25]. ORing describes routing where each node, except for the originator and the target node, knows only its predecessor and successor. Determining the original source and target node is therefore difficult. ORing was introduced in the 1990s to ensure privacy.

We now describe the establishment of an ORing connection. The visualization can be observed in Figure 2.1. Public-key cryptography is used. The originator, in this case A, chooses a list of nodes from which it creates a circuit between itself and the target node D. A creates a circuit consisting of itself and of nodes B, C and D. A creates a fixed-sized cell encrypted in layers using the respective public keys of the nodes in the circuit. The cell contains addresses of the circuit nodes and a different symmetric session key (SSK) for each node. A stores the SSKs and sends this cell to B. B removes the outer layer of the cell using its private key in order to get the address of its successor, in this case C, and the SSK assigned to B. B stores the SSK. B cannot remove any additional layer because it lacks the private keys

2. The Bible is restricted in some countries, e.g. North Korea, or Iran [26].

of the successive nodes. *B* sends the cell to *C*. *C* removes yet another layer, stores the received SSK and, sends the cell to *D*. *D* removes the final layer and stores its SSK. Now *D* is able to communicate with *A* through *B* and *C* using symmetric cryptography.

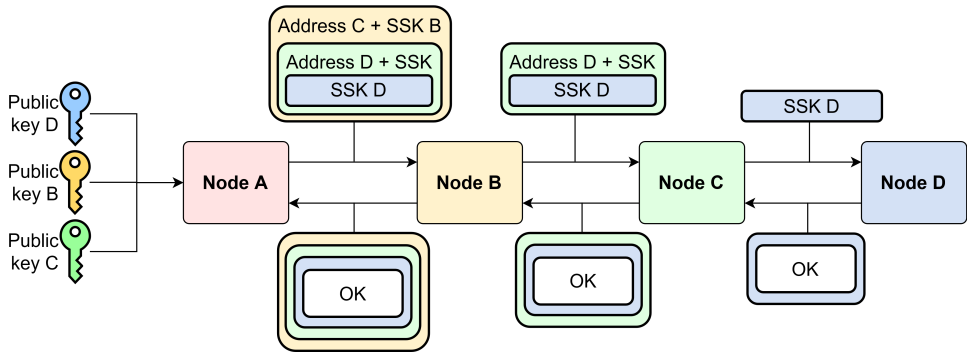

Figure 2.1: This is how a connection is established in ORing. The individual steps are described in Subsection 2.2.2.

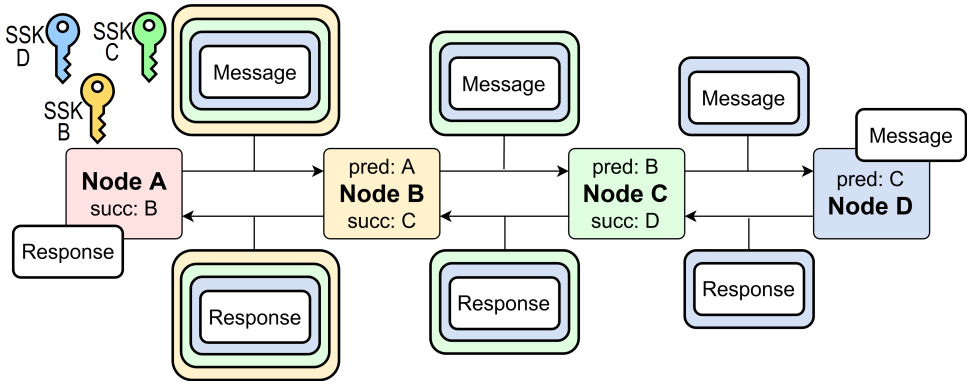

Figure 2.2: The communication between nodes in ORing is visualized in this image. The individual steps are described in Subsection 2.2.2.

The communication in ORing is described in Figure 2.2. Node *A* encrypts a message using the SSKs of *D*, *C*, and *B* in this order into a fixed-sized cell. It then sends the cell to *B*. *B* removes one layer of the cell, this time using its stored SSK. The cell is sent to *C* and *D*

in a similar manner. *D* encrypts its response into a fixed-sized cell leveraging its stored SSK. The cell is sent to *C*. *C* encrypts the cell utilizing its SSK. The cell is sent to *B* and *A* in a similar manner. Node *A* decrypts the cell using all three SSKs.

2.2.3 I2P

I2P [1] is another network of the dark web which anonymizes its traffic. It is accessible through the I2P browser. In contrast to Tor, communication in I2P is based on *garlic routing* [2]. Garlic routing is described as an extension of ORing. The established tunnels between two nodes are unidirectional, meaning different tunnels are used for outgoing and incoming messages. Another difference from ORing is the bundling of messages. An individual message is called a *clove*[3]. Each clove contains its own delivery instructions. These instructions are exposed at the target node. Cloves are bundled into a garlic-message. The bundling of cloves ensures more secure and efficient communication.

Both exampled technologies provide anonymous access to the clear web as well as to the dark web. Both are open-source and free to use.

2.3 The data set

The dark web pages were acquired via web scraping as part of a bachelor's thesis [65]. The scraped pages belonged to two different networks - I2P and Tor. The total number of pages was 221,844. Of those pages 212,851 were Tor pages and 8,993 I2P pages. The total number of unique domains was 5,178 of which 4,912 were from the Tor network and 266 were from the I2P network.

The fields of a page entry are described in the list 2.3. Each description includes a concrete example from the database. Fields beginning with an underscore are assigned to every document implicitly.

_id is a unique identifier. For example *2d622b6fba6f203d790fedbb4f47963e2366c7fd*.

3. Michael Freedman, who defined garlic routing, called cloves *bulbs*.

_index informs about the collection the document belongs to. For example *tor*.

_type determines the type of the document. For example *_doc*.

content is the actual content of the page. For example *Purple Kush – 10g – WackyWeed Menu * Home * Contact us * About us* .

content_type describes the type of the content. For example *text/html; charset=UTF-8*.

domain of the url address. For example *wacky2yx73r2bjys.onion*.

h1 is the text with the h1 style. For example *Purple Kush – 10g*.

links to other pages this page links to. For example *{*
"link": "http://wacky2yx73r2bjys.onion/",
"link_name": "Home"
}.

raw_text is similar to *content*. It additionally may contain formatting elements such as *\n*. For example *Purple Kush – 10g – WackyWeed Menu\n\n * Home\n * Contact us\n * About us\n* .

raw_title is similar to title. It additionally may contain formatting elements. For example *Purple Kush – 10g – WackyWeed*

raw_url is the same as *url*.

title of the page. For example *Purple Kush – 10g – WackyWeed*.

updated_on depicts the time when the document in the database was last updated. For example *2019-10-22T19:41:09*.

url address of the page. For example *http://wacky2yx73r2bjys.onion/?product=purple-kush-10g*.

2.4 Ahmia

Ahmia is a search engine mainly for domains of the Tor and I2P networks. Ahmia disposes of crawlers. The Ahmia crawlers were leveraged for the scraping of the pages. This is described in the bachelor's thesis by Juraj Noge [65]. The mentioned thesis was being finished at the same time as this thesis.

2.5 Elasticsearch

The data collected from the dark web was stored in Elasticsearch. ES is a distributed, open source search engine [13] and offers a fast full-text search. Another benefit of ES are documentation and supported tools, such as Logstash [12] for processing of data or Kibana [14] for the visualization of data. ES can be used as a NoSQL database. Such a database consists of indexes, documents and fields as opposed to tables, rows and columns of SQL databases. One of the advantages of NoSQL databases is scalability, therefore they are suited for big amounts of data.

3 Classification

A part of this thesis was the categorization of the scraped pages. Categorization in this context means the procedure of assigning categories to pages. The data set at hand contained a vast amount of pages, as we discussed in Chapter 2. The manual categorization of this number of pages would therefore take too much time and in consequence was infeasible. An automatic system for text classification was required. A simple system with a list of words assigned to each label (naive approach) was introduced but was performing poorly. This naive approach is outlined in Section 3.1. A more sophisticated tool for text categorization is machine learning. We decided to use supervised machine learning (ML).

This chapter describes the naive approach. Secondly ML in general is characterized. Then, the main approaches of ML, reinforcement learning(RLr), unsupervised learning (ULr), and supervised learning (SLr), are characterized. Lastly we depict what an artificial neural network (ANN) is. The approach used in this thesis is a type of an ANN using SLr.

3.1 Naive approach

The naive approach first leveraged for the classification on a sample of about 4,000 pages was a two dimensional (2D) list[1] with words. The first dimension represented the categories. The second dimension was a list filled with words frequently associated with the category the list was dedicated to. For example the list for the category *Drugs* contained among others the words *weed, marihuana, cannabis, cocaine, coke, hashish, heroine, grams, drugs, ecstasy*. Such a list was created for each category. The categories will be described in more detail in Chapter 5.1.2. Next, the words in the content of the pages were normalized and the twenty most frequent words of each page were compared to the 2D list. Each conformity was counted and the results were stored in a list as a category - conformity-count pair. The

1. With 2D list we mean a list of lists.

category with the highest conformity-count was then assigned to the page. The accuracy of this approach was below 40%.

The 2D list was not exhaustive. In order to include all the relevant words for every category we would need to study the data set extensively. This in unviable in the context of the data load as we discussed earlier in this chapter. Also, special meanings of specific words would not be taken into consideration. For example the words *red room* have a special meaning. This naive approach could not detect such connections. In light of the listed problems we decided to not pursue this technique further.

3.2 Machine learning

The term *machine learning* was first introduced by Arthur Samuel. In his paper *Some Studies in Machine Learning Using the Game of Checkers* [73] he proved it is possible for a program to develop better game-related skills than the skills of the programmer of the program.

ML is the procedure in which an agent[2] creates a model which develops the ability to perform a certain task. The learning process is based on the evaluation of gained experience [46]. The agent requires an initial data set in order to train and validate. The size of the initial data set depends on the nature of the task and the selected ML type and set of algorithms. During training the agent attempts to perform the given task and validates its own performance. The agent then adjusts its criteria for performing the task based on the validation results. These two steps are repeated a number of times defined by the programmer. The output of the agent is a model able to perform the task it was trained for.

ML is used across various industries and fields. The need for automated analysis is increasing with the growing popularity of *big data*[3] [81] [7]. ML is used widely with Big Data. ML is also used in the language processing field [46]. Examples include virtual assistants or instant translation tools. Another use for ML is in the automotive industry. Cars with assisted parking or breaking, or self-driving cars

2. A system utilizing ML to learn and adapt autonomously [72]
3. A data set too vast or complex for traditional or manual data processing.

are examples[41]. Also, ML is used frequently in games for a more natural game experience [28].

There are several approaches in ML based on the different ways of training. We describe the three main approaches in the subsections following. Namely *reinforcement learning* in Subsection 3.2.1, *unsupervised learning* in Subsection 3.2.2, and *supervised learning* in Subsection 3.2.3. The approach used in this thesis was *supervised learning*.

3.2.1 Reinforcement learning

It is suitable to use RLr if behaviours in dynamic environments are to be learned [48]. A software agent receives an indication of the state of the environment. The agent then pics an action from a discrete set of agent actions. Next the state is modified by the action. The value of this modification is passed on as a scalar reinforcement signal to the agent. The objective for the agent is to maximize the long-run total of reinforcement signal values. This is achieved over time by leveraging a number of specialized algorithms together with methodical trial and error. RLr is well suited for example for the development of game-behaviour [47].

3.2.2 Unsupervised learning

ULr is used when seeking common patterns or relationships in data. It is also commonly used when trying to find anomalies in data [21]. The software agent receives an unlabeled data set as input. The agent breaks the data down to critical components using specialized algorithms. It then identifies similarities and divides the data into groups. No feedback is involved. One of the fields ULr is used is astronomy. For example for tasks such as estimating the photometric redshifts or finding galaxy clusters [40].

3.2.3 Supervised learning

SLr is suitable to use for the classification of data. During training SLr relies on labeled data [72]. The input of the software agent of SLr is a labeled data set. The software agent divides the data—label pairs into a training and test set randomly. The agent is to find a function with

the data as input and the correct label as output. To achieve this the agent is trained on the training set and validated using the test set. After each run the accuracy and loss is computed and the function is modified with the goal to improve the future output. SLr is suited for example for the classification of data [64].

3.3 Artificial neural networks

An ANN is a ML system inspired by animal brains [72]. A brain is composed of components such as neurons. Neurons communicate with each other by passing information through synapses. The more often two neurons communicate the stronger the synapses between them are [35]. This results in a neuron prioritizing information passed by neurons participating in frequent communication.

A structure of an ANN is portrayed in Figure 3.1. An ANN comprises layers of nodes sometimes also called *neurons*. The neurons communicate via directed links of various relevance also called *weight*. The weight is represented by a real number. A neuron determines whether to react to information gathered through a link according to its weight passed to an *activation function*. We discuss activation functions (AcF) later in Subsection 3.3.5. Neurons exist in groups called *layers*. Layers used in this thesis were *embedding layers*, *convolutional layers*, *polling layers*, and *dense layers*. All of the used layers are described closer in the next subsections. Layers between the input and output layers are called *hidden layers*. If an ANN contains convolutional layers it is called a convolutional neural network (CNN).

One learning cycle is called an *epoch*. In case of big data sets the data are divided into smaller chunks of data. These chunks are called *mini-batches*. In this paragraph we characterize a typical cycle over a mini-batches (batch-cycle) in an ANN using SLr. A single epoch consists of at least one batch-cycle. At the beginning of a batch-cycle the training data are passed to the first layer. Data are passed from one layer to the next one. Every layer modifies the received data before passing it on. The manner in which the data are modified depends on the type of the layer. After the data are processed by the last layer the output model is evaluated using the testing data. Im-

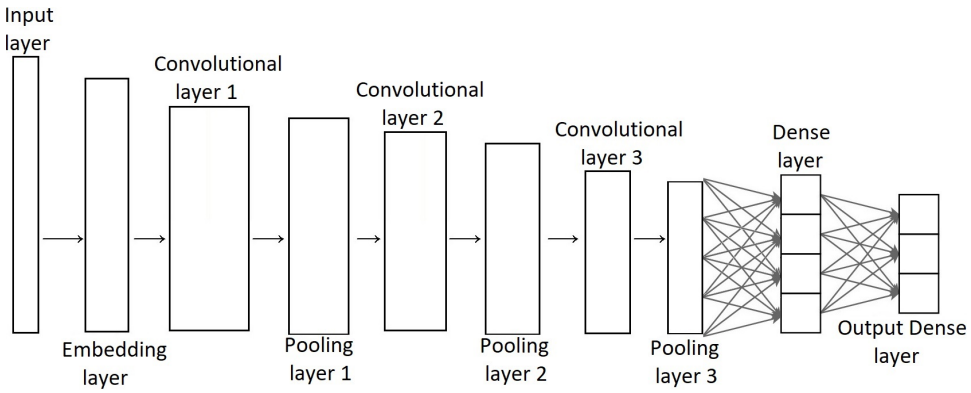

Figure 3.1: An illustration of the CNN implemented in this thesis. The two portrayed dense layers are a modified image from an article on deep learning [16]. The individual layers are detailed in Section 3.3.

proving the model is now the objective. A way leveraged to improve accuracy is first the calculation of the loss by utilizing a *loss function*. The loss function is outlined in more detail in Subsection 3.3.6. After evaluating the loss the weights of the neurons are adjusted by the *backward propagation of errors* detailed in Subsection 3.3.7. After the weights of the first layer have been adjusted the batch-cycle ends. The epoch ends after all batch-cycles finish. The number of batch-cycles in an epoch depends on the number of mini-batches. The number of batch-cycles and epochs is specified by the programmer.

3.3.1 Embedding layer

Embeddings [53] compose a matrix of word aliases representing how semantically meaningful relations between words are. For example the relation between the words *queen* and *princess* is in some way similar to the relation between the words *mother* and *daughter*. An ANN operating on words achieves good results if some sort of embedding is adopted [55]. An embedding layer (EmbLr) creates an embedding matrix via training. The matrix is generated by assigning embedding vectors to each word in a vocabulary of all words in the input. The distance between the vectors of two words describes the relations be-

tween them. From the vectors of the previous example the result vector could imply $queen - princess = is\ predecessor\ of$.

The EmbLr may also use a pre-trained embedding matrix. The layer then passes the input words replaced by the embedding vectors to the next layer.

3.3.2 Convolutional layer

Convolutional layers (ConvL) are suited for finding features in data [5]. A ConvL decreases the size and complexity of the input by convoluting the input. The convolution of data are a way of combining two sets of data. It does so by sliding a *filter*, also called kernel, of a certain size with a certain stride across the input and performing a *convolution operation* (ConvO). An example of a simple ConvO is visualized in Figure 3.2. The stride size represents the number of input-units, e.g. pixels, the kernel needs to skip in order to perform the next ConvO[4]. The kernel is represented by real numbers and is initialized with random numbers. A ConvO can have multiple kernels. Each kernel is applied to a portion of the image corresponding to the kernel size, called the *receptive field* [59]. This action is called a (ConvO). After applying the kernel to all possible receptive fields the output of the layer is generally smaller in size than the input. It is, however, possible for the output dimensions to be the same as the input dimensions. Moreover, the information carried in the borders of the input may not be lost. This is useful when the number of ConcLs is rather sizable. It is achieved by applying *padding*. Padding is the practice of adding information around the borders of the input, e.g. *zero-padding*[5]. The result of the convolution is fed to a specified AcF. AcF is detailed in the later Subsection 3.3.5.

The first ConvL implemented in this thesis receives as its input the normalized and encoded content of the scraped pages. The other ConvLs receive their input modified by other layers. The ConvLs then

4. The sliding direction is to the right. If sliding to the right is not possible the filter is returned to the very left and stride-size down. If sliding down is not viable the sliding is finished.

5. Zero-padding is a specific type of padding. The input is enriched with zeros around the borders.

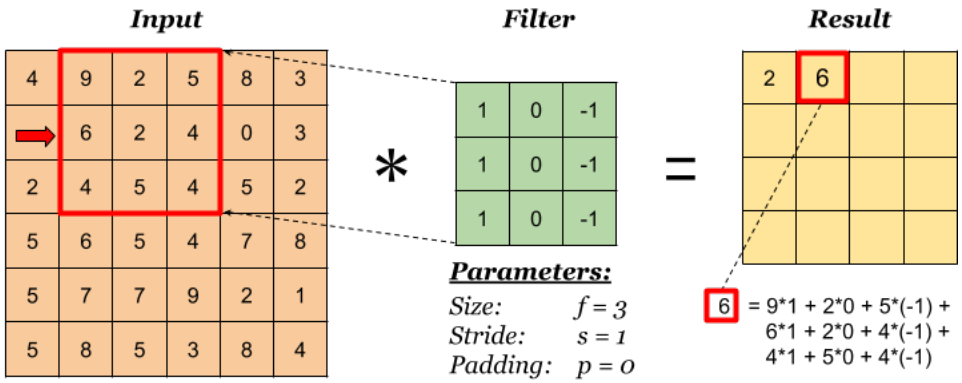

Figure 3.2: The visualized process of a ConvO [44]. The very left square is the input – a 2D image of size 6x6 pixels. The square in the center depicts a 3x3 kernel with no padding. The stride of the kernel is 1 pixel. The very right square represents the partial result after the kernel was applied to two sections of the input. The final output will be of size 4x4 pixels. The ConvO itself is enumerated under the output image.

proceed as described earlier. In doing so the size of the output compared to the input is reduced. Also, features of the input significant to the layer are highlighted.

3.3.3 Pooling layer

Pooling layers (PoolL) reduce the size of the input by down-sampling. The procedure resembles a sliding window of a certain size sliding over the input with a certain stride. The stride in this layer is similar to the stride in ConvLs mentioned in Subsection 3.3.2. The result of each step of the pooling depends on the method used. There are several types of methods to choose from for the PooL. One of the most used type is *max-pooling* [5]. The result of max-pooling is the maximum value from the values currently present in the filter window. After applying the filter to all input portions the result is passed to the next layer.

A simple example case of max-pooling is shown in Figure 3.3. In the beginning the upper left corner of the filter is set to be in the upper

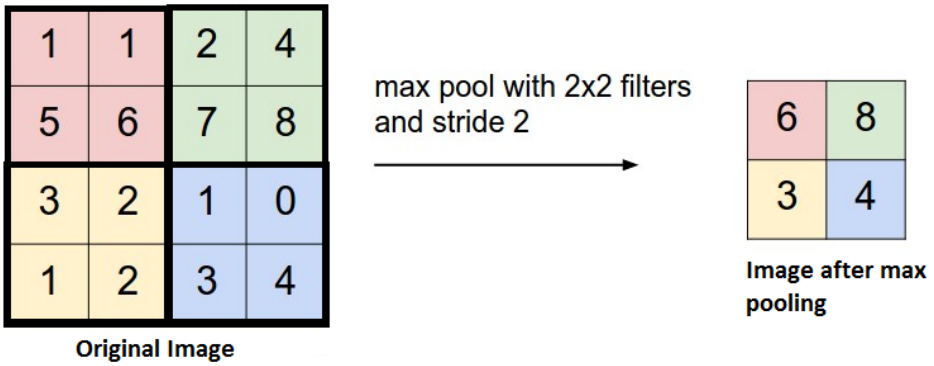

Figure 3.3: The visualized process of max-pooling with filter size of 2x2 pixels and stride length of 2 pixels by Analytics Vidhya [6]. The input is a 2D image and is depicted by a square with scalar values. The size of the image is 4x4 pixels. The square on the right is the result after max-pooling is applied on the input. The output image is of size 2x2 pixels. The max-pooling process is described in Subsection 3.3.3.

left corner of the input which is the red part of the image. The maximum value from the values *1; 1; 5; 6;* is *6*. The filter therefore returns the red result *6*. Next the upper left corner of the filter moves two pixels to the right. It now occupies the green part of the input image. It again performs the max-pooling operation. It is not possible for the filter to slide right anymore as the end of the input was reached. The filter is therefore returned to the beginning of the line and moved two pixels down into the yellow portion. After returning the yellow result the filter slides two pixels to the right and once more return the maximum value. It is not possible to move the filter right nor down. The Max-pooling now is finished an the result is the square on the right.

The PoolLs implemented in this thesis receive as their input the output of the foregoing ConvL. The layer applies max-pooling. In doing so the size of the next input is again reduced by selecting the maximum features.

3.3.4 Dense layer

Each neuron in a dense layer (DnsL) is directly connected to all the neurons from the previous layer as well as to all neurons in the next layer [5]. DnsLs are therefore also called fully connected layers. DnsLs are well suited for classification tasks. A DnsL is visualized in Figure 3.4. The advantage of a DnsL is every output of every neuron of the previous layer is processed. The outcome z_n^l for neuron n^l is enumerated as a multiplication of two matrices w and x with the formula

$$z_n^l = w_n^l x^l + b_n^l \text{ [78].}$$

Where n^l is a node belonging to layer l. Weight w_n^l are all weights of direct connections into neuron n^l and x are all inputs of said connections into neuron n^l. The notation b_n^l symbolizes the bias of node n^l. The result of the operation described is fed to a specified AcF. AcF is detailed in Subsection 3.3.5. The main disadvantage of DnsLs is the number of input arguments.

The first DnsL implemented in this thesis receives as its input the output of the last pooling layer. It then proceeds as described earlier. The second DnsL receives its input from the first DnsL. This layer ensures the classification of the data.

3.3.5 Activation function

An AcF is used to modify the output of a neuron most commonly in a non-linear way in order to limit the amplitude of the output of a neuron [49]. The output of an activation function is called *unit's activation* or just activation. There are many types of AcFs. In this thesis we used the AcF called *rectified linear unit* (ReLU) and *softmax*.

ReLU is an AcF used in multilayer networks [39]. The function itself is shown in Figure 3.5. It is expressed as

$$\sigma(x) = \begin{cases} 0, & \text{for } x \leq 0 \\ x, & \text{for } x > 0 \end{cases} \text{ [39]}$$

where σ represents ReLU. ReLU is used for its reduction of activation since only positive input values are having a non-zero output.

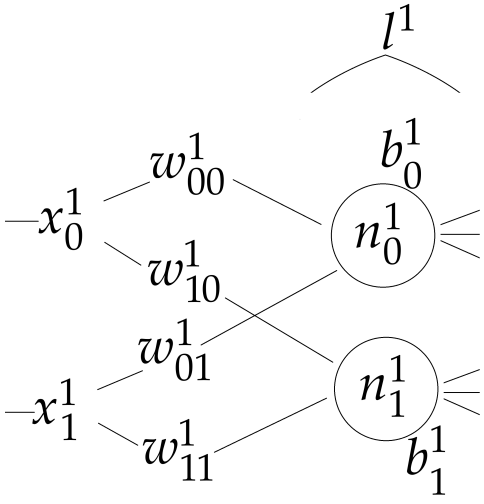

Figure 3.4: The visible portion of this network consists of a DnsL l^1. This layer is composed of two neurons, n_0^1 and n_1^1 and their corresponding biases b_0^1 and b_1^1. Each neuron receives two inputs x_0^1 and x_1^1 with weights w_{00}^1, w_{01}^1, w_{10}^1, and w_{11}^1 respectively. The operation the neurons of the DnsL perform is now described on neuron n_0^1. Neuron n_0^1 computes the outcome z_{00}^1 as $w_n^1 \cdot x^1 + b_0^1 = \begin{pmatrix} w_{00}^1 & w_{01}^1 \end{pmatrix} \cdot \begin{pmatrix} x_0^1 \\ x_1^1 \end{pmatrix} + b_0^1$ which is broken down to $w_{00}^1 \cdot x_0^1 + w_{01}^1 \cdot x_1^1 + b_0^1$.

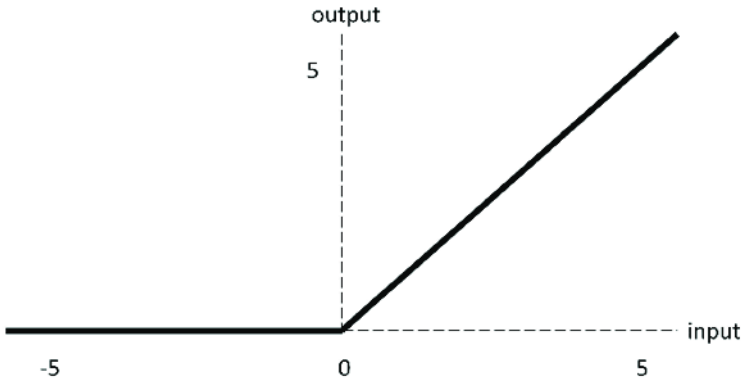

Figure 3.5: The ReLU AcF from the Multi-classification of Brain Tumor Images using Deep Neural Network [38] article.

Another advantage is information of positive input values is not lost.

Softmax is an AcF leveraged for the designation of probabilities of an input belonging to beforehand given labels [72]. The activation

is a vector of numbers each between 0 and 1. Each number represents the probability of the input belonging to the current label. In our case labels were categories of pages.

3.3.6 Loss function

] As was discussed in Section 3.3, an ANN needs to learn in order to improve results. To be able to claim the results have improved an objective measure needs to be introduced. *Utility* represents such a measure [72]. By utility we mean the total satisfaction of the learning agent. A way to obtain the amount of utilities lost is to utilize a loss function (LsF). Most generally, a LsF receives an input x, the actual label of the input y and the predicted label $\hat{y}$. The result of the LsF is computed as

$$LsF(x, y, \hat{y}) = Utility(\text{result of using } y \text{ given an input } x) \\ -Utility(\text{result using } \hat{y} \text{ given an input } x) \text{ [72]}$$

There are several LsFs. The one we used is called *categorical crossentropy* (CCross).

CCross is used when the task is to label inputs and the amount of available labels is more than two [45]. Also, it must be possible to assign each label with the likeliness of it belonging to the given input. The formula for CCross loss computation is

$$-\sum_{c=1}^{M} y_{i,c} log(p_{i,c}) \text{ [45]}$$

where M symbolizes the total number of labels and c is the predicted label[6].The symbol $y_{i,c}$ denotes a binary indicator[7] signaling whether c is the correct label for sample i. The symbol $p_{i,c}$ stands for the predicted probability that sample i belongs to label c.

3.3.7 Backward propagation of error

The ouputs of individual layers need to improve in order to improve the results of an ANN. It is rather simple to determine the loss of the

6. Labels are represented by scalar values. Labels are converted if they were of non-scalar format, such as text.
7. An indicator which holds one of two exclusive values, such as 0 and 1 .

last layer utilizing the LsF. However, it is not possible to determine the loss of hidden layers in the same way. Therefore the error from the last layer needs to be propagated to the previous layers. This procedure is called backward propagation of error (back-prop) [72].

The weights and filters of individual layers are modified so that after the next forward-propagation[8] the LsF output is smaller. The way the weights and filters are modified depends on various factors, e.g. the type of the layer, the AcF used, or the position the layer is in.

In case mini-batches are used back-prop alone is not efficient in minimizing the output of the LsF. Therefore optimizer algorithms are introduced into the back-prop. The goal of optimizers is to make the output of the LsF converge to the global minimum of the LsF in a smaller number of epochs. The optimizer used in this thesis is RMSprop. RMSprop updates the weights dynamically based on the history of the weight values.

8. The flow of the data in the direction from the input layer to the output layer as opposed to backward-propagation.

4 Clustering

One of the goals of this thesis was to visualize the structure of the scraped portion of the dark web. We chose to render the data as a web graph which is further outlined in Section 4.1. As was described in Chapter 2, the data set to be displayed was rather sizable. The problem with the visualization of such an amount of data is readability. All pages of such a data set cannot be displayed at the same time if additional information, such as the category with the url address of the page, needs to be provided to the user. Such rendering would be cluttered and readability would be affected negatively. It was therefore necessary to find a proper way to divide the graph into several subgraphs. We decided to adopt community structure introduced in Section 4.1.1. For community detection we leveraged two algorithms for comparison purposes. One of them is the well known Louvain algorithm (LA). LA is outlined in more detail in Section 4.2. The other one is the Leiden algorithm (LeA), an improved version of LA. LeA is described in the Section 4.3 following LA.

In this chapter we characterize web graphs and their challenges. Next we talk about community structure and LA, the algorithm for dividing a graph into communities.

4.1 Web graph

The data are displayed as a web graph [37]. A web graph is a graph representation of the web. Nodes are portrayals of the pages and edges depict links between the pages. Web graphs tend to be built from an enormous amount of data. As such, they can be advertised in various ways. One of the visualizations is shown in Figure 4.1. The depicted web graph displays all its data at once without any labels or details. The result may be useful for viewing the internet as a whole. However, for our purposes this view was not sufficient. One of the goals of this thesis was the possibility of the inspection of the relationships between nodes in more detail. The big amount of data described in Subsection 2.3 prevented the displaying of all pages at once. The information the user would read from such a graph would be either incompatible with the requirements or

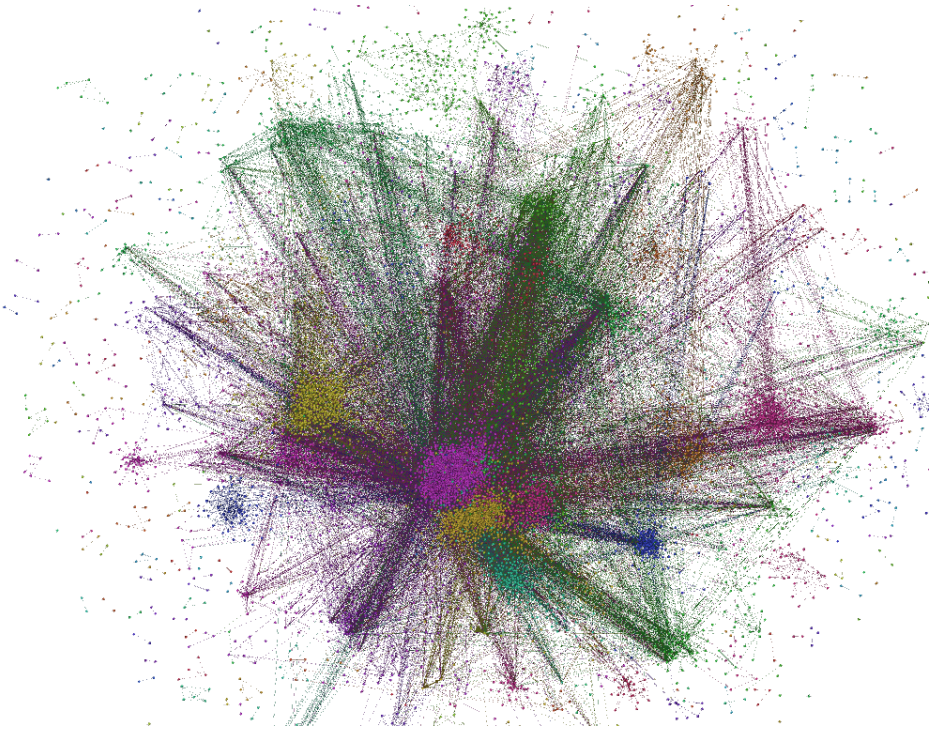

Figure 4.1: Web graph by Citeo made consisting of circa 600 000 domains and 16 billion links. [20].

incomprehensible as too much visual data worsens the quality of the user's information processing [60]. Therefore the requirement arose for the graph to be composed of a significantly smaller amount of nodes. As a result the user would be able to obtain the sought knowledge without hindrance. A considerable amount[1] of nodes in our graph were not isolated[2]. Those nodes were in fact part of a single connected component. It was therefore possible to partition the graph based on the density of its nodes into communities. Communities are characterized in detail in the next Subsection 4.1.1.

1. Out of 210,191 node 90,275 were connected and 119,916 were isolated.
2. An isolated node is a node with zero incoming and outgoing edges.

4.1.1 Community structure

If a graph can be partitioned into several subgraphs so that nodes from one subgraph are internally connected densely and are connected scarcely to nodes from other subgraphs, we can claim it has a community structure. Each subgraph of such a graph is a community [29]. Each community can be portrayed as a meta node of the graph. This way the number of nodes in the graph is reduced. The quality of such a partition is measured using modularity. L. Wenye and D. Schuurmans describe modularity in their work [63] with the following words:

> For a candidate partition of the vertices into clusters, the modularity is defined to be the portion of the edge connections within the same cluster minus the expected portion if the connections were distributed randomly [56].

Modularity is represented by a number between -1 and 1. If the value is positive the connections between nodes of the same cluster are more densely connected than the randomly distributed connections between the same nodes.

4.2 Louvain algorithm

A widely used algorithm for finding communities in graphs is LA [10]. It is a greedy algorithm which maximizes modularity locally. In this algorithm, modularity is an indicator of the density of connections between nodes belonging into the same communities as opposed to links between communities. The modularity calculation is also taking into account whether the graph is weighted[3] or not.

Next we detail the principle of the algorithm. We example each step on node N_A belonging to a portion of an example graph illustrated in Figure 4.2.

a) Each node is assigned a community. Current modularity for each node is calculated.

3. A graphs in which the links have wights assigned to them.

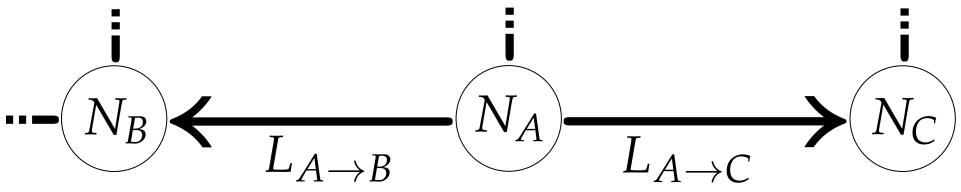

Figure 4.2: The visible portion of the exampled graph depicts nodes N_A, N_B, and node N_C. There are links $L_{A\to B}$ and $L_{A\to C}$ between nodes N_A and N_B, and N_A and N_C respectively. Other links are displayed partially and are connecting the nodes to the rest of the graph.

- Node N_A is assigned to community C_A. Let us assume modularity $M_{A\to cA}$ for N_A is *0.2*.

b) Each node is disassociated from its community and randomly appointed to a community of one of its neighbours. This is repeated for each neighbour of the node. Modularity for each node after each such a transition is calculated.

- Node N_A has two neighbouring nodes, node N_B in the community C_B and node N_C in the community C_C. Node N_A is removed from C_A and appointed to C_B. Let us say modularity $M_{A\to cB}$ is *-0.1*. Afterwards, N_A is removed from C_B and assigned to C_C. Let us suppose modularity $M_{A\to cC}$ is *0.5*.

c) Each node is now appointed to the community in which the maximum modularity was achieved. This can also result in the node remaining in its original community.

- The maximum modularity of N_A achieved in the previous step is $M_{A\to cC}$. N_A is therefore removed from C_A and appointed to C_C.

d) Discard the empty communities.

- Community C_A is now empty and is therefore discarded.

e) Each community is now considered a node (community-node). Links between community-nodes are constructed from links of nodes of the same community-node (old-links). This is done by grouping together old-links which target nodes assigned to the same target community-node. These grouped links now represent weighted edges between community-nodes. Old links between nodes of the same community-node are represented by a self-loop on the community-node.

   - Community C_C is now considered a node N_{cC} and C_B is considered a node N_{cB}. A link $L_{cC \to cB}$ from N_{cC} to N_{cB} with a weight of 1 is created because of link $L_{A \to B}$. Also, a loop $L_{cC \to cC}$ is created on N_{cC} because of the link $L_{A \to C}$. The result of this step can be observed in Figure 4.3.

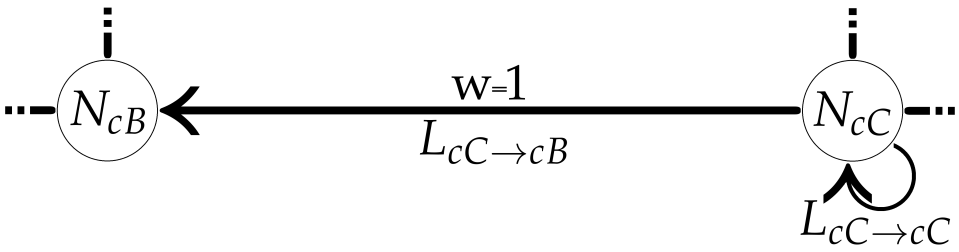

Figure 4.3: This graph is the result of applying the above listed steps to the previously shown portion of the example graph from Figure 4.2. The visible portion of the graph contains nodes N_{cB} and N_{cC}, and a link $L_{cC \to cB}$ with a weight of 1 between them. A self-loop $L_{cC \to cC}$ on N_{cC} is present.

The above detailed steps are repeated until modularity is not improved any more.

LA is favoured for its simplicity, speed and accuracy. Since its introduction, in 2008, it was possible to detect communities in graphs with billions of nodes in a relatively timely manner. LA was compared to other algorithms for community detection [10]. Namely the algorithm of Wakita and Tsurumi [79], of Pons and Latapy [68],

and of Clauset, Newman and Moore [17]. The used graphs were of sizes varying between 34 nodes and 77 edges to as much as 118 million nodes and 1 billion edges. The difference between the computing times of the previously stated algorithms grows with the size of the graphs and favours LA. In fact, it took 152 minutes for LA to detect the communities of the greatest graph whereas the computation time of the other algorithms was more than 24 hours. In terms of precision, LA was also the most precise one with slightly better modularities.

4.3 Leiden algorithm

LeA is another algorithm for community detection on large graphs. The authors of LeA claim LA to have a major flaw [77]. LA may detect badly connected or internally disconnected communities[4]. The latter phenomenon occurs when a bridge-node[5] of one community is assigned to another community and the remaining nodes are not. LeA is based on LA and eliminates the before mentioned problems of LA.

Next we outline the principle of the LeA. The detailed steps are portrayed in Figure 4.4 [57]. The steps are compared to the steps depicted in the characterization of LA in Section 4.2 and described using the visualization.

a) This phase corresponds to step LA a).

   - *(Figure 4.4) a)*. Each node is assigned a community. This is portrayed by the nodes being of different colours.

b) This phase is called **move nodes** and corresponds to steps *LA b)*, *LA c)*, and *LA d)*.

   - *(Figure 4.4) b)*. The nodes are divided into three communities represented by the colours of the nodes - a red, a blue and a green one.

4. It is not possible to form a connected component from nodes of an internally disconnected community.
5. A node connecting two or more connected components otherwise not connected between each other.

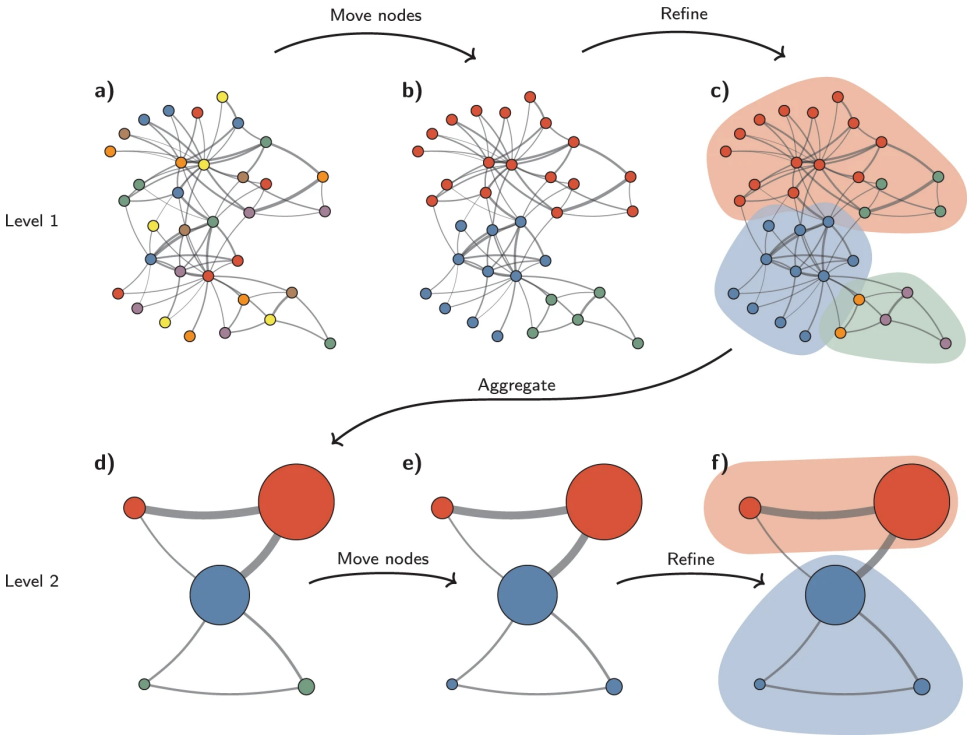

Figure 4.4: The visualization of the principle of LeA [57] by the authors of *From Louvain to Leiden: guaranteeing well-connected communities* [77]. Steps *a)* to *d)* are described in the above included characterization. Steps *e)* and *f)* show a part of a second iteration of the algorithm. The algorithm ended in step *f)* because no further improvement was achieved by **refining** the communities.

c) This phase is called **refine**. Steps *LA a)* and *LA b)* are applied to nodes within communities. This results into the further partitioning of communities into sub-communities.

   - *(Figure 4.4) c)*. The communities are refined within communities. The red community was refined into two sub-communities - a red-red one and a red-green one. The blue community was not refined further. The green was refined into a green-range one and a green-purple one.

d) This phase is called **aggregation** and corresponds to step *LA e)* but sub-communities are treated as communities. Nodes created from sub-communities within one community are considered to belong to the same community. This consideration is taken into account in further iterations. Otherwise, these nodes are regarded as individual nodes. This practice prevents bridge-nodes to be appointed to a different community and so disconnect the former community.

   - *(Figure 4.4) d)*. The sub-communities are considered nodes now. However, sub-communities from the same community are still considered to belong together. Therefore the colours of the nodes corresponds to the colours of the communities from the previous step.

These steps are repeated until no further improvement in modularity can be achieved.

Step d) is in further iterations significant. It is impossible to reassign a bridge-node to a different community with LeA. But this may happen using LA. Now we demonstrate why on Figure 4.4 *part f)*.

A bridge node in the blue community connecting the two smaller nodes is depicted. Let us call this bridge-node *B*. With LeA, all blue nodes are still considered to be of the same community. And therefore they cannot be shifted into any neighbouring communities. Let us now assume we ran another iteration of LA over the graph in Figure 4.4 *f)*. The blue nodes are not considered belonging to the same community anymore. The nodes are viewed as unrelated neighbouring nodes. There is therefore a possibility for node *B* to be assigned to the red community if higher modularity is achieved. This would leave the two smaller blue nodes disconnected.

In this thesis versions of both LA and LeA were used.

5 Development

An application was developed and a CNN was trained in order to fulfill the goals of this thesis. One task of the application was to classify the scraped pages into categories. This was to be done depending on the content of the pages. The application performs the classification via the model created by the CNN. CNNs and ANNs are described in Section 3.3. Another task was to provide a way to observe the structure of the pages, links between them, and their categories. This was achieved by detecting communities of a graph. Communities and how to detect them are described in Chapter 4. One of the partner requirements was for the application to function on a UNIX system. Another requirement was a user friendly UI. Also, the option for retrieval all the available information about the pages was demanded.

The architecture of the application is portrayed in Figure 5.1. This chapter first describes the implementation of the model for the classification of the pages. Afterwards the implementation of the clustering is introduced. Then the design and implementation of the application which is composed of a representational state transfer (REST) application program interface (API) and a front-end (FE) web application is detailed.

5.1 Classification

To categorize the pages depending on their content a CNN needed to be created. The number and the structure of the available data are detailed in Section 2.3. A project called `Categorization` was built. For the training of a CNN a suitable training and testing data set needed to be prepared. The desired output of the CNN was a model able to categorize the pages with a reasonable accuracy. This chapter describes the steps taken in order to achieve the listed goals along with the technologies used.

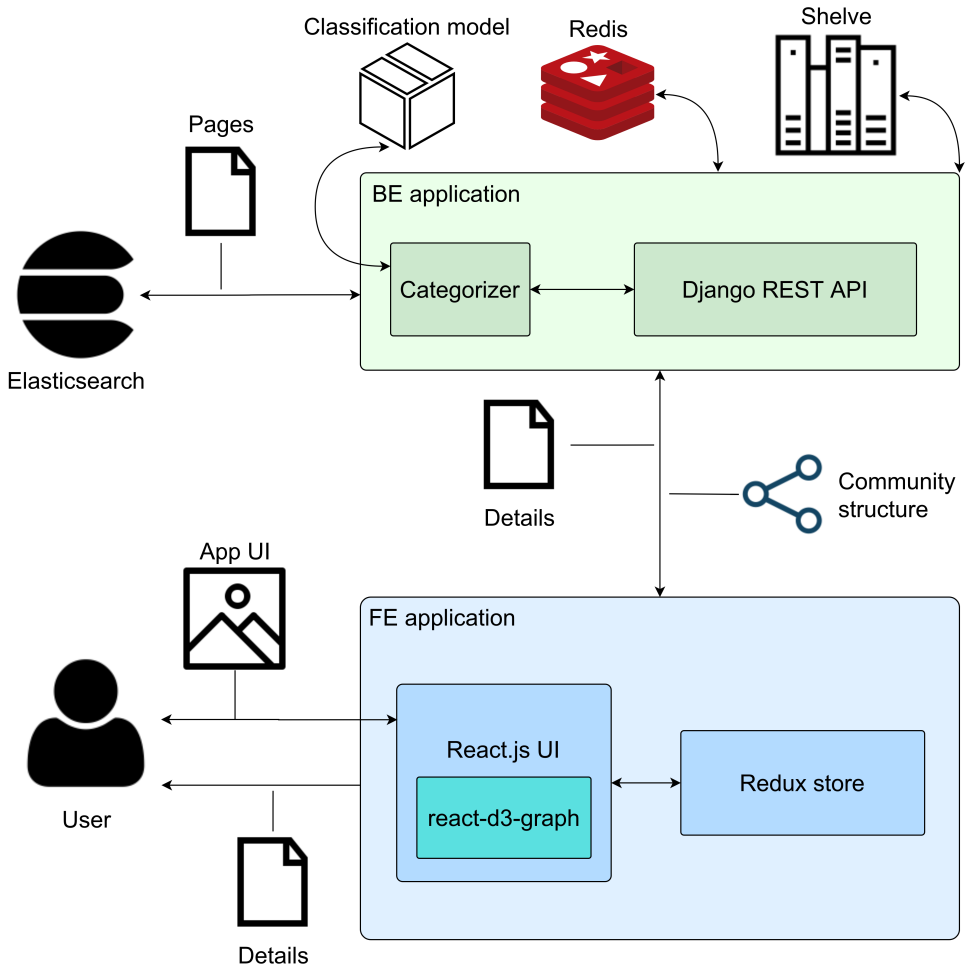

Figure 5.1: The visualization of the architecture of the web application.

5.1.1 Technology overview

The project was written in *Python*. *Python* is a widely used interpreted programming language known for readability and portability [33]. It is open-source and is considered to have an extensive documentation and community available. *Python* is popular in the science community because it is easy to learn and has simple syntax. There is therefore a considerable amount of useful libraries for research purposes

such as *Keras* described later in this subsection, or *igraph* introduced in Subsection 5.2.1.

The learning approach used in this thesis was SLr introduced in Subsection 3.2.3. A sample data set with assigned categories was therefore needed. To view and edit a data set with thousands of rows the software *EmEditor* [27] was leveraged. *EmEditor* is a paid text editor with a free trial period. The advantage of this editor is its support of large files. Supported file formats include comma-separated values (CSV) files.

For the modification of the content of the pages we utilized the library *Natural Language Toolkit* (NLTK) [70]. *NLTK* is a *Python* library used for operating on human language data. *NLTK* provides functionality such as tokenization, or stemming. The library supports several languages including English. This library comes with detailed documentation. The classification of this thesis supports only English.

For the management of large matrices and vectors we used the library *NumPy* [24]. *NumPy* is a *Python* library used for scientific computing. The functionality includes for example the formation of n-dimensional array objects, the random shuffling of rows of such an object, and random number capabilities.

To manipulate the data set efficiently the *pandas* library [18] was adopted. *Pandas* is an open-source library used for the manipulation of data frame[1] objects. It is possible to slice through, add data to, or remove data from the data frames. Also, data frames may be merged or reshaped. Supported file formats include CSV and excel spreadsheet (XLS) files.

The CNN itself was created with the library *Keras* [50]. *Keras* is a library for the implementation of ML in *Python*. ML is more closely characterized in the Section 3.2. *Keras* supports the tokenization of text and the tools needed to build and train a CNN. The output model of the CNN can be used for the task it was trained for.

1. A data frame is a two-dimensional structure resembling a common table. Data stored in columns contains values designated for the same purpose.

5.1.2 Learning data set

A labeled data set for learning was needed. We therefore built a learning data set composed of a subset of the scraped pages. The pages were retrieved from the database with equivalent fetching functionality as is described in Subsection 5.3.2. This functionality is located in the folder `helpers`. Only one page with content of each domain from the Tor network was stored. 4,088 pages were obtained this way. These pages were with content exported into a CSV file and labeled manually. 13 categories were identified. We will call this learning data set *first samples* in a later chapter. The problem of this data set was some categories contained less than 100 pages. The training result for those categories were not satisfying. Detailed training results performed on various training data sets are shown in Section 6.2.

By the merging of categories with less than 100 pages into bigger categories we achieved better training results. One category, *Gambling*, could not be merged with other categories as it was not related enough to any other category. We therefore enhanced Gambling with more pages from the database. This was possible because we found a domain with more than 200 pages with different content all associated to gambling. This data set contained 4.298 pages and 10 categories. We will further call this learning set *enhanced samples*. However, a moral issue related to the naming conventions of the categories arose with this data set.

The final data set was created by further merging and renaming the categories. 9 categories were identified. We will call this data set *final samples*. The 9 categories detected are detailed as follows:

Finance and Fraud contained 665 pages. It included content about fake or stolen credit, debit and gift cards and bank accounts. Also crypto currency, counterfeit money, and money laundering was mentioned. Suspicious and illegal investment opportunities were offered as well.

Gambling contained 231 pages. Content involving casinos, bets and other form of gambling constituted this category.

Hosting and Programming and Hacking contained 411 pages. Content assigned with this category involved technical blogs

and hosting servers. Hacking tutorials and services were also offered.

Illegal services and goods contained 139 pages. Provided services involved the hiring of hitmen[2] and fake identity services. Offered goods included drugs and guns.

Online Marketplace contained 143 pages. Content labeled with this category consisted of mentioning products of other categories along with electronics, clothing and accessories. All of these goods were offered at once on one page.

Other contained 806 pages. Pages in this category were not possible to categorize. The content was either not English, too incoherent, short, or vague.

Sexual Content contained 1,196 pages. Content in this category was of a sexual nature including sexual stories and the mentioning of images or videos.

Social contained 136 pages. This content included blogs, chat rooms and information channels. Also censored content was found, such as the Bible or confidential information. Other content found was written works of art and mentions about paintings and music.

5.1.3 Implementation

The preprocessing of the learning data and the training of the CNN takes place in the file `keras_classification_model.ipynb`.

The learning data are loaded as a data frame. The data preprocessing is performed on the content of the pages. Non-English rows, also called documents, are removed. This is done by first removing non-numeric and non-alphabetical characters. Then the content is tokenized[3]. In this case the tokens represent words divided by spaces. The tokens are then compared to the English vocabulary of NLTK. According to the percentage of the English words composing the content

2. A hitman is person hired to kill another person or people.
3. A tool which divides a string into sub-strings.

the document is considered English or non-English. The threshold of English documents is 20%. This threshold was chosen after discussions with the supervisor. Non-English documents are removed from the data frame.

Next the labels need to be preprocessed. The labels are retrieved from the data frame by removing duplicates from the category column. Then a dictionary[4] with the category name as key and the index[5] as value is created. The indexes are used as category aliases.

Afterwards two empty lists are initialized, `texts` and `labels`. For each row of the data frame the content is appended to list `texts`, the category is converted to the matching alias and appended to the list `labels`. The text is associated with the analogous label via the indexes in the lists. This is explained on an example. The row on position i contains content T_i and the corresponding category C_i. Content T_i is appended to `texts` and is on the ith position. The alias for category C_i is C_i^a. Now C_i^a is appended to `labels` and is also on the ith position.

Next an internal vocabulary is initialized from all words from `texts`. The list `texts` corresponds to a matrix. The matrix is stored in a list of matrices called `data`. The list `data` is split into a training and testing set. One fifth of `data`, meaning 864 entries, is randomly chosen to be the test set. The rest are the training data.

Each category alias in `labels` is assigned an index i. Each entry of the list is replaced by a one hot vector (OHV)[6] . The value 1 is on the ith position of the OHV. The length of the OHV coincides to the number of categories. The list `labels` becomes a matrix after this modification, meaning a list of vectors.

The next step is the configuration of embeddings. Embeddings are outlined in Subsection 3.3.1. An open source embedding matrix is available on the Stanford university web site [67]. However, using that matrix changed the learning results only marginally as opposed to training the embedding layer on our data. One of the reasons might be the vocabulary and the content syntax of the dark web being different from the vocabulary used in the trained embeddings file. We

4. A python dictionary is a hash table with keys and values.
5. An index in context of this chapter is an integer value corresponding to the position of an entry. It starts from 0.
6. A zero vector with one value being 1.

therefore decided not to use the trained embeddings. An example of an embedding vector is enclosed in the Appendix 10.1.1.

The last step before the training itself is to create the shape of the CNN. The *Keras* documentation offers a sample of a CNN structure meant for text classification [51]. We used this structure and Figure 3.1 depicts the structure composition.

The EmbLr receives information about the vocabulary size, the number of words on each page and the length of the output vectors. The vocabulary is limited to 20,000 most frequently occurring words. The number of words in the content of each page is limited to 1,000. The EmbLr therefore expects an input of vectors with the length 1,000. The output of this layer are embedding vectors with the length of 100.

The second layer is a 1D ConvLr with 128 neurons and a kernel with the dimensions 5x1. The activation function of this layer is ReLU described in Subsection 3.3.5. Next a PoolLr with filter dimensions of 5x1 is prepared. The method used in the pooling layer is max-pooling. The next layer is a 1D ConvLr followed by a PoolLr and another 1D ConvLr all equivalent to the first layers of the corresponding types. The seventh layer is a PoolLr also using the max-pooling method but with the filter dimensions equal to the input dimensions. Then a dense layer with 128 neurons and ReLU as the activation function follows. The last layer is a dense layer with the activation function softmax. The number of neurons in this layer is the number of categories.

Afterwards a *Keras* model is fed the CNN structure and compiled. The configured loss function of the model is CCross. The desired metrics to be evaluated is the accuracy of the results. The optimizer used is RMSprop. The model is then trained with training and testing data. The number of epochs is set to 20 and batch_size is 128.

After the training finished the model was exported. It then was used in the (back-end) BE for the categorization of the pages.

5.2 Clustering

The structure of the pages was to be visualized in the form of a graph. The amount of the pages described in Section 2.3 needed to be divided into groups in order to view the structure without the loss of

information. For this purpose both LA and LeA were used separately. These algorithms are characterized in Section 4.2 and Section 4.3 respectively.

5.2.1 Technology overview

In the beginning we used the library *NetworkX* [23] for the graph creation. *NetworkX* is a *Python* library for the managing of complex networks. The advantage of this library is convenient documentation. Also, the graphs of *NetworkX* cover many characteristics such as undirected and directed, or weighted edges. It is possible to filter isolates from the graph. Another advantage is the availability of standard graph algorithms, including LA. However, the built in LA implementation was too slow for the size of our data set. It didn't succeed to divide about 40,000 pages after 30 minutes. We therefore decided to use a different implementation. The *cylouvain* library [31] was introduced.

Cylouvain is a faster implementation of LA. The division of 90,275 pages into communities took about 15 seconds. The problem we faced with this implementation was the occasional occurrence of disconnected sub-communities of communities.

Our new priority was therefore to adopt an implementation of LeA. The *leidenalg* [76] library was introduced. *Leidenalg* is a *C++* implementation of the LeA exposed to *Python*. This implementation takes about 6 seconds to partition the 90,275 pages into communities. Moreover, there are no disconnected sub-communities of communities. *Leidenalg* is built on the *python-igraph* (igraph) library.

Igraph is another library for the managing of complex networks. The advantage of this library the availability of the before mentioned implementation of the LeA.

The two utilized implementations are compared in the Section 6.1.

5.2.2 Implementation

The clustering is implemented in the BE application. The application is further described in the next Section 5.3. We describe the clustering in this subsection.

The clustering itself (cluster cycle) is executed in the method `get_groups_without_links_and_isolates` in the `graph_helpers.py` file. The entry parameter are either scraped pages or communities. Both will act as nodes. The implementations of the clustering algorithms mentioned in the previous Subsection 5.2.1 require the nodes to be in a compatible Graph format.

First, the pages need to be converted into integer aliases. The aliases will constitute the vertices of the graph. The page links also need to be in the form of integer aliases. These link aliases will compose the edges of the graph. For the graph initialization one of the libraries *igraph* or *NetworkX* is utilized. The decision is made based on a boolean flag whether to use LeA or LA. Now the graph is be filled with the edges and the vertices. Next, isolates are filtered out and deleted from the graph vertices. Afterwards the communities are detected, the result is a partition. This partition is a dictionary with the node aliases as keys and the communities assigned to the nodes as values. At last the aliases in the dictionary are converted back to the original node *ids*[7]. The method returns the partitioned node ids and the isolates if any were detected.

The clustering cycle is repeated on communities until the number of detected communities is not greater than the desired maximum number (max count). This is demonstrated in Figure 5.2. Another reason to end the repetition is if the number of detected communities cannot be reduced any further. Max count in this thesis is 50.

5.3 API

The scraped pages of the dark-web were being stored in *ElasticSearch*. We created a BE in order to perform various operations on the dataset before sending it to the FE. Such operations involve resource intensive processing of large volumes of data, and caching. We decided to initialize a BE utilizing *Python*. The reason behind this decision was the requirement for the application to function on a UNIX system. Other reasons are described in Subsection 5.1.1. The requirements for the BE was the categorization of the nodes along with their division

7. *Ids* in case of pages are urls. *Ids* in case of communities are specific strings, e.g. *2.14.0*.

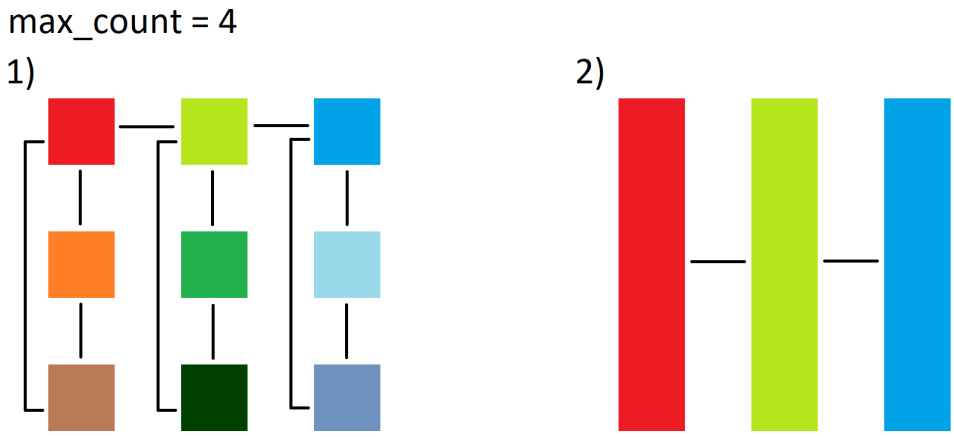

Figure 5.2: An example of multiple cluster cycles. Max count is four. The image on the left represents a hypothetical output of the first cluster cycle. It contains 9 squares each of a different colour with undirectional links between them. Each square depicts a community. The number of the communities is bigger than max_count. Therefore the cluster cycle is repeated with the 9 communities as the nodes to be clustered. The second cluster cycle detects three communities portrayed as rectangles in the right image. This number satisfies the max count condition and the cluster cycle is not repeated.

into groups. The groups were either represented by nodes of the same category or communities. The BE also needed functionality to return details for specific pages or groups if requested.

5.3.1 Technology overview

The BE was written in *Python*. As we wanted to follow the REST architecture we decided to make use of the *Django* framework [30]. *Django* is responsible for tasks such as running the server or managing web requests. Another advantage of *Django* is its *Django REST framework* (DRF). *DRF* offers a convenient way for creating restful endpoints and responses [58]. Both frameworks are open-source with helpful documentation and community.

We had to solve performance issues. It took approximately 30 minutes to retrieve and categorize about 220,314 pages from the database and circa 13 seconds to divide such a response into communities. We considered this wait time to be too long. Therefore we decided to cache the data of the first response. For that purpose *Redis* [54] was used.

Redis is an open-source solution which we used as a key-value store. It supports basic data structures[8] as values, but not custom objects. Since the API uses custom objects for *communities*, *pages*, or *links*, an object serializer was leveraged along with *Redis*. We decided not to write our own but to utilize the *Python pickle* module[9] [32] (pickle). The reason behind this decision was the simplicity of *pickle*. *Pickle* also fulfilled all our needs for serializing. Specifically the serialization or deserialization of data in the form of the previously mentioned models for *Redis* to store in an acceptable time.

Pickle took about 6 seconds to serialize 220,314 pages. The deserialization of the same number of pages took about 7 seconds. However, the combination of *pickle* and *Redis* was not stable enough for the storing of sizable objects. The acquisition of page content was one of the tasks needed of this thesis. As page content was a sizable object we decided to utilize the *Python* module *shelve* [34] (shelve). *Shelve* manages *shelfs*. A *shelf* file is a persistent *Python* dictionary. The advantages of using *shelve* are the possibility of storing large objects and no need for serialization of keys or values. *Shelve* took approximately 61 seconds to retrieve 220,314 pages with content.

5.3.2 Implementation

This API handles four use-cases. One of which handles the acquisition of pages divided into communities depending solely on links between the pages. The next use-case manages the community detection based on categories. This means on the highest the pages are divided into groups solely by page category. On lower levels the pages are divided into communities depending on the links between the

8. Simple structures, e.g. strings, numbers or sets.
9. A module used for converting *Python* objects to streams of bytes and vice versa.

pages. The third use-case is the retrieving of further details of a specified page. The last use-case is the retrieval of group details[10].

The BE is implemented in the project `Dark-web-categorization-BE`. The project was initialized in the `category` folder. The `settings.py` and `wsgi.py` files were included in the *Django* boilerplate. File `urls.py` contains the endpoints for the whole application. The urls relevant to this thesis are from the `api` module and imported into this file. The `api` module is situated in the `api` folder. The implementation relevant for this thesis is situated here. We now detail the most important files.

`view_sets` contains ViewSets[11] of the API.

`urls.py` registers routes with corresponding *ViewSets* to a *Django router*[12].

`apps.py` holds the boilerplate configuration file for the API.

`classification` module contains the categorization functionality.

- **`labels.py`** lists the available categories with the corresponding indexes.
- **`categorizer.py`** encompasses the class `Categorizer` used for the categorization in the BE. Implementation details are included in Appendix 10.2.1.

`models.py` accommodates the *Django* models[13]. The models and the relationships between them are visualized in Appendix 10.2.2.

`serializers.py` contains serializers[14]. Each model used in the client-server communication has a matching serializer assigned. An example of handeling a response is detailed in Appendix 10.2.3.

---

10. Page details of all pages belonging into a specified community.
11. A ViewSet is a class which provides responses according to the received parameters. The request methods *GET* and *POST* are supported.
12. An object which assigns ViewSets to the corresponding endpoints.
13. A class containing the fields and behaviour of data.
14. A serializer transforms *Django* models into different formats, e.g. JSON.

`utils` module encompasses various helper methods. These methods are leveraged for example for the partitioning of the graph, caching or page detail gathering.

`repositories.py` holds the methods which handle the fetching from the database. More information about the methods in this file are described in Appendix 10.2.4

5.4 Front-end

In order to display the data acquired from the BE in a reasonable way a FE application was created. The goal of the FE was to visualize the scraped pages in a graph. The pages or communities, and links from the BE were to depict nodes, and links of the graph respectively. The categories of pages or communities needed to be readable in the graph. Additional information about the nodes needed to be displayed or retrieved on demand.

5.4.1 User interface

We designed the UI the following way.

The UI is composed of a *header* with the name of the application *Dark web categorization*, a *graph* and a *sidebar* as can be observed in Figure 5.3.

The graph-nodes represent either communities pages are partitioned into, or the pages themselves. Both are exampled in FigureMixedGraphBasic.

Community nodes are depicted as *pie charts*. The individual sectors of the *pie chart* represent the categories of the pages belonging into the community.

Page nodes are portrayed as square shaped symbols. The colour of the square corresponds to the category of the page.

Community-nodes can be double-clicked. After doing so, a new graph is shown. The data of this graph consists of the sub-communities of the clicked community. We call this process *zooming*. If further zooming is not possible, the user reached the

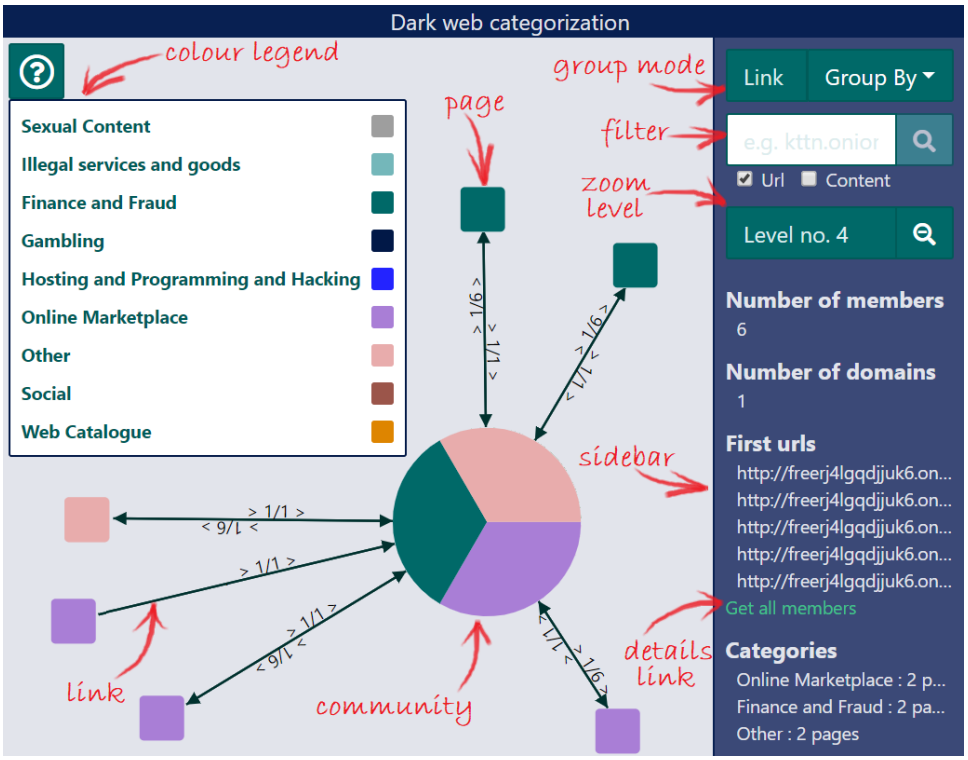

Figure 5.3: The application UI. The red arrows and text are explanatory and not part of the UI.

maximum zooming level. Each community may have a different maximum level, depending on the number of its pages and its structure. If a community at the maximum level has no siblings it is displayed in one of two ways. The community is decomposed into page-nodes if it contains at most 30 members[15]. We call these communities decomposable. Otherwise it is portrayed as a community-node.

If the graph contains any isolated nodes (isolates) a mock-community is displayed containing all isolates. This community cannot be zoomed into.

15. Community members are pages belonging into the community.

A button with a question mark is positioned in the upper left corner of the UI. If clicked, a legend with the available categories and their colours is shown. The legend is labeled *colour legend* in Figure 5.3.

Arrows between nodes symbolize the links between pages or communities as is illustrated in Figure 5.3. Each link has a text in the form of >*x*/*y*> assigned to it. Where x is the number of links of the source node with a common destination. And y is the total number of links of the source node. The direction is given by the > symbols. For instance, the exampled link in Figure 5.3 indicates the lila page linking to the community with one out of a total of one links. Bidirectional links have two indicators assigned.

At the very top of the *sidebar* a *drop-down button* (DD) is present. It is labeled as *group mode* in Figure 5.3. The DD sets the mode according to which the pages are grouped into. There are two modes available, *link-mode* and *category-mode*. If *link-mode* is selected, the pages are divided into communities. If *category-mode* is selected, the pages are divided into groups by categories. Zooming into such a group results in its pages being further divided into communities as if in *link-mode*.

Underneath, a filter consisting of an *input field* with a *submit button* was placed. The filter is also visible in Figure 5.3. The purpose of this element is the filtering of nodes according to a search phrase.

An indicator is placed below the filter. The indicator marks the current level - how many times the user zoomed into a community. It is labeled as *zoom level* in Figure 5.3. The indicator has a *zoom-out button* placed next to it. After this button is clicked, the user is shown the communities of the previous level.

After single-clicking a node additional information is exampled in the *sidebar*.The details vary depending on whether the node is representing a community or a single page. The details of an individual page are as follows:

Url which also serves as a unique identificator of the page.

Category of the page. Each page belongs to exactly one category.

Linked pages are url addresses to other pages. There are up to 30 links visible in the sidebar.

The details of a community are visible in Figure 5.3. They consist of grouped information of its members and include the following:

Number of members represents the number of all the pages belonging into the community.

Number of domains is the sum of all domains in the community.

First urls addresses (urls) of members belonging into the community. There are up to 30 urls visible in the sidebar.

Categories are the categories of all the pages of the community. Each category is represented by its name and the number of pages belonging into this category.

A link for the acquisition of further node information is present in the *sidebar*. The link is labeled as *details link* in Figure 5.3. If clicked, a pop-up window with detail-options is displayed. Details may include the *title*, *category*, page *content* and *links*, depending on the user's selection. *Urls* of the page or pages are present by default. The pop-up window can be observed in Figure 5.4. After selecting the options and clicking the *download button*, a *JSON file* with the desired information is downloaded.

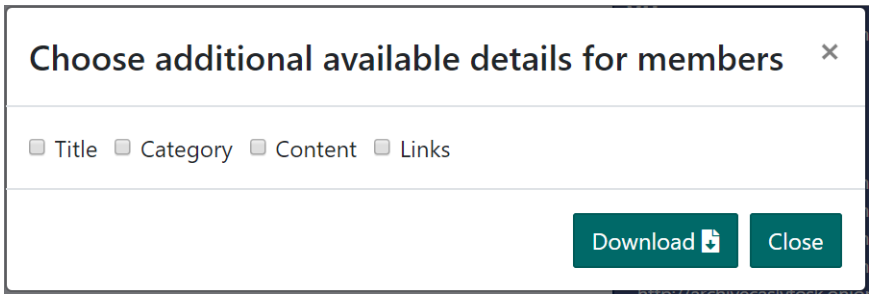

Figure 5.4: The pop-up window with detail-options for selection. These options dictate which details will be included in the downloaded text file.

5.4.2 Technology overview

JavaScript [62] (JS) is the most favoured programming language used for creating web applications [82]. It is an interpreted language supported by all modern browsers. It is open-source and as such disposes of a considerable community with convenient documentation. *JS* is not strongly typed and the code might therefore be complicated to read or navigate. For this reason the FE was written in *TypeScript* (TS) [61] which is a superset of *JS* with the advantage of being typed. There is a significant number of tools and libraries for implementing user interfaces (UI) in a clean and timely manner created for both *JS* and *TS*. One of such tools is the framework *React.js* [42] (React). *React* is one of the most favoured *JS* frameworks [43]. The advantages of using *React* are readable code and improved performance by managing the re-rendering of page elements.

To achieve a satisfying UX the app needed to be interactive and obtain new or modified data frequently. Repeated requests to the BE would mean longer wait time for the user. A proper mechanism for data-storing would present a convenient solution to this issue and make the requests unnecessary. A *JS* library which handles the app state and works well with *TS* and *React* is called *Redux.js* [3] (Redux). *Redux* is a single store approach. This ensures easy hydration[16]. *Redux* also provides a custom set of *TS* typings and provides the developers with easy to use debugging tools. Another advantage of this library is the documentation.

The visual aspect is implemented using Less [75] which is a language extending *CSS* with improvements such as the possibility of using variables.

The visualization of the web graph alone was realized using the *react-d3-graph* library [15] (RD3). *RD3* is an implementation of the library *D3.js* [11] made more convenient for the use with *React*.

5.4.3 Implementation

The FE project consists of three folders and several configuration files. This subsection describes the most important folders in the `src` folder.

16. The process of filling the attributes of an object with values.

`_customized_react-d3-graph` contains the customized react-d3-graph module [15] written by Daniel Caldas. A portion of the modifications were written by Antonin Klopp-Tosser in an open pull-request [52].

We enhanced the module so that arrow-heads pointing to custom shaped nodes are visible. This was achieved by determining the direction vector of the link, normalizing it and then multiplying the vector by the shortest distance between the borders of the source and the target node.

`models` contains models and their interfaces. Each file contains one server-model and one client-model. The conversion between these models is conducted in specific helper functions. The advantage of this approach is the independence of client-models from the BE models.

`styles` holds the *Less* classes. The classes are divided into files according on the element they are meant to modify.

`utils` folders enclose helper functions such as converters between server and client models.

`constants` folders hold string constants or simple functions which return a string depending on the input.

`components` folders contain files with *React* components. Components define the skeleton of the UI with the specified behaviour.

`containers` folders hold files with *React* containers. A container is a file with access to the app-state. It is responsible for passing data to components.

`@types` encompasses custom typings for libraries which do not provide own typings. Typings for popular libraries are often downloadable as modules.

The remaining folders represent some part of the state modification by the *Redux* framework. We enclose the structure of the remaining folders in Appendix 10.3.2 as it complies to the standard *Redux* file structure.

6 Evaluation

6.1 Clustering results

We compared the two used algorithms LeA and LA. The results are visualized in Appendix 9.

The LA detected 10,072 communities of which 9,915 were decomposable. The decomposable communities contained 33,600 pages in total. The number of disconnected sub-communities was 31. The discovering of all 10,072 communities took 59.7 seconds. The time it took to detect the communities on the highest level was about 14.7 seconds.

The LeA detected a 7,857 communities of which 7702 were decomposable. A total of 26,372 pages comprised the decomposable communities. No disconnected sub-communities were found. It took the LeA 17 seconds to detect all 7,857 communities. The highest level was partitioned in 5.2 seconds.

We valued the number of decomposable communities detected by the LA. However, the disconnected sub-communities presented a problem in the correctness of the graph structure. Also, the LeA was about three times faster in detecting the communities. The algorithm used for the API responses is therefore the LeA.

We decided to adopt LeA for the visualization of the pages on the FE.

6.2 Classification results

Four labeled data sets were created as was described in Section 5.1.2. Ten independent models were trained on each data set. Another ten models were trained with pretrained embeddings on the final samples. The accuracy results of these models are depicted in this section via box plots[1] [80].

The lower and upper sides of a box in the plot represent the first and the third quartile respectively. The horizontal line inside the box portrays the median. The plus symbol in the box is the mean. The lower and upper whisker ends represent the minimum

1. A visual tool for the imaged summarization and comparison of data.

and maximum values respectively. The name of each box plot corresponds to the utilized training sample data set. Table 6.1 pairs the category shortcuts used in the plots with the actual category names.

Alias	Category
C1	Art
C2	Deviancy
C3	Drugs
C4	Encyklopedia and knowledge
C5	Fake identity and hitmen
C6	Finance and Fraud
C7	Gambling
C8	Guns
C9	Hosting, Programming and Hacking
C10	Online Marketplace
C11	Other
C12	Porn
C13	Social
C14	Web Catalogue
C15	Illegal services and goods
C16	Sexual Content

Table 6.1: The alias-category lookup for the figures in Section 6.2.

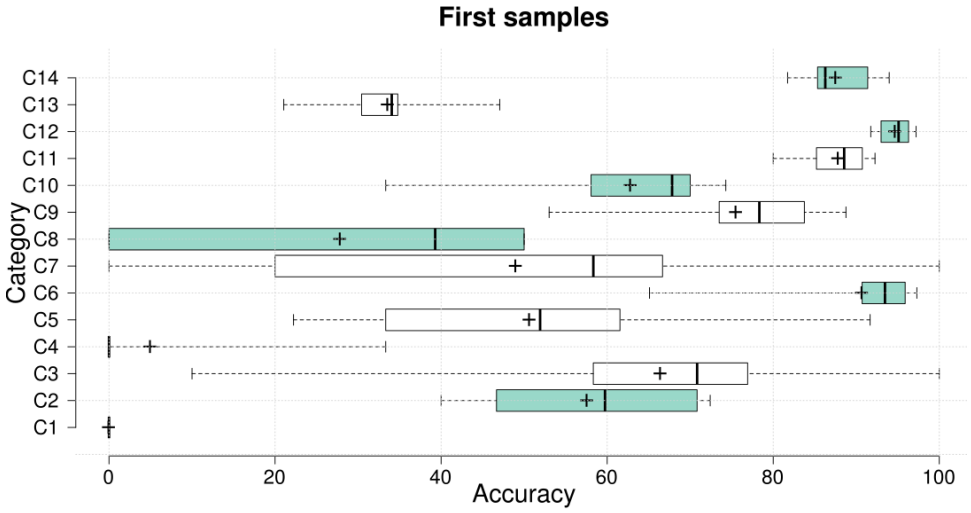

Figure 6.1: The accuracy per category of the models trained on the first samples. The median of several categories of the first samples were below 70%. The dispersion was often over 50%. The individual key values of this plot are detailed in Appendix 8.1.

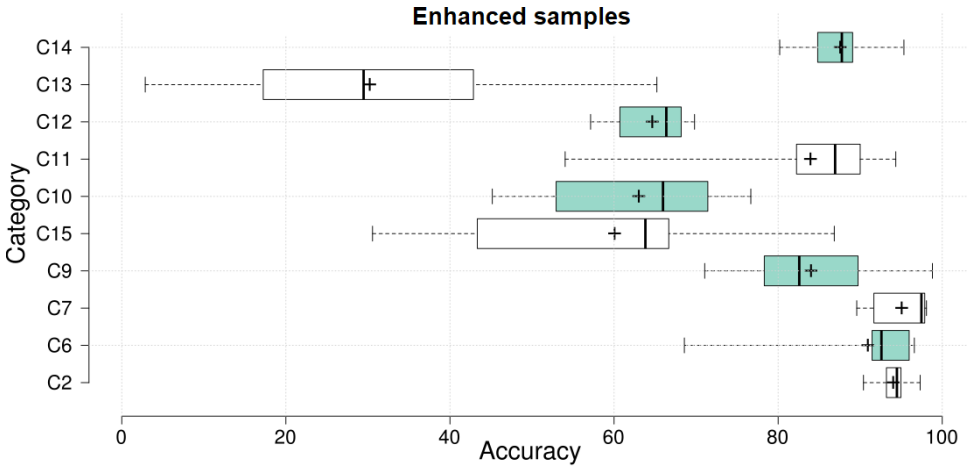

Figure 6.2: The accuracy per category of the models trained on the enhanced samples. The median of most categories was over 65%. The dispersion was lowered. The individual key values of this plot are detailed in Appendix 8.2.

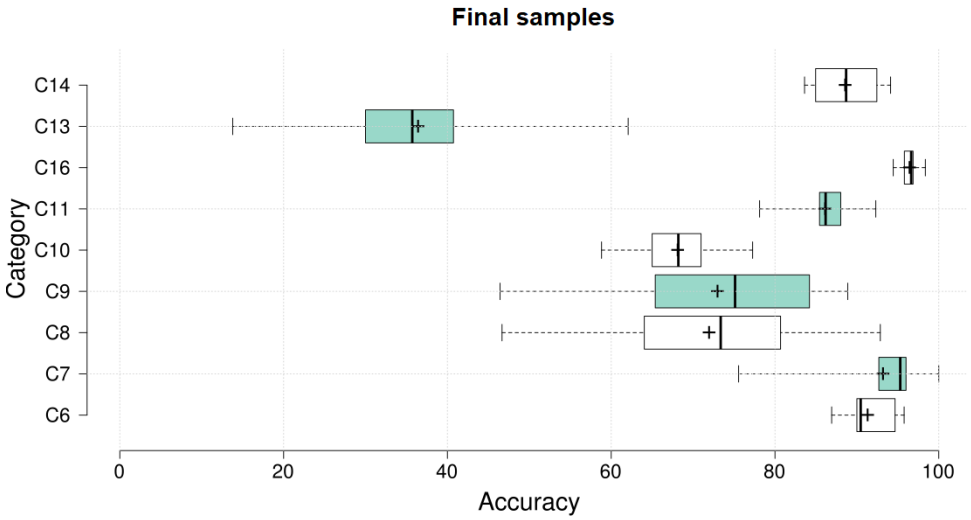

Figure 6.3: The accuracy per category of the models trained on the final samples. The dispersion was low compared to the results in the enhanced samples. The individual key values of this plot are detailed in Appendix 8.3.

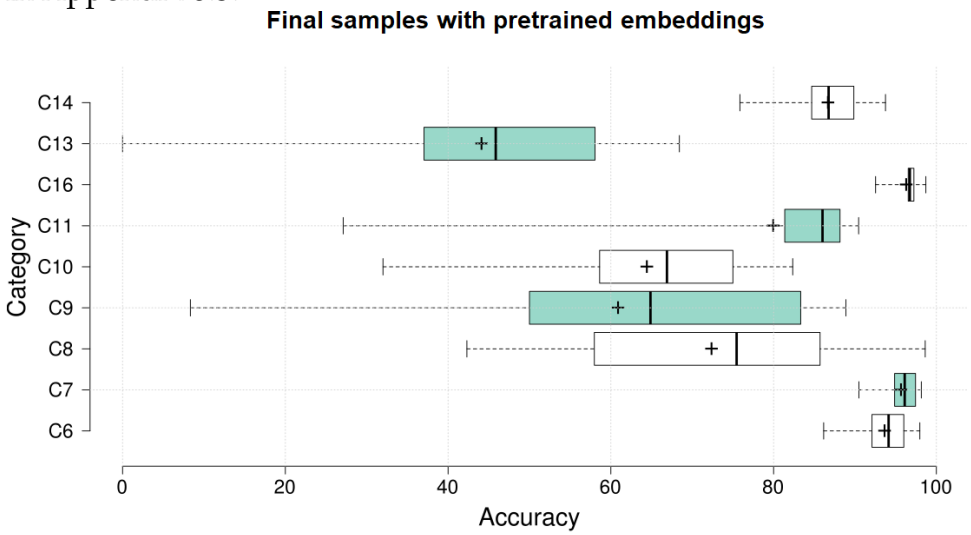

Figure 6.4: The accuracy per category of the models trained on the final samples with pretrained embeddings. The maximums and the third quartiles were better compared to the final samples without pretrained embeddings. However, the dispersion of the results was comparatively as bad as in the results with the first samples. The individual key values of this plot are detailed in Appendix 8.4.

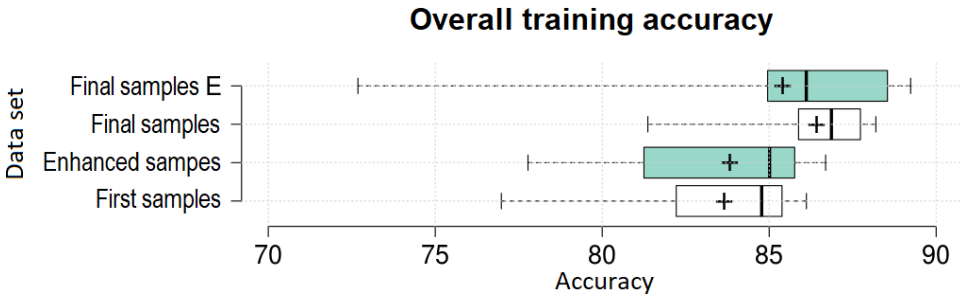

Figure 6.5: The overall accuracy of the models trained on the dataset samples described in Section 5.1.2. The data set *Final samples E* stands for the training on the final samples with pretrained embeddings. The individual key values of box plots are detailed in Appendix 8.5

The learning data set used for the model training was the final samples. The final model utilized in the BE for the categorization of the pages achieved 88.2% accuracy. We compared these results to two other projects both from the year 2017. The ATOL [36] classification tool scored an accuracy of 96.4%. ATOL was used for the categorization of the dark web into 3 categories - drugs, hacker, weapons. The learning data set contained 529 sites.

The second classifying tool [4] we compared our results to accomplished an accuracy of 96.6%. This tool classified the data into 9 categories. The learing approach chosen was SLr. The learning data set contained 6831 pages.

Both other projects have achieved a higher accuracy. However, ATOL recognized only 3 divergent categories which may explain the high accuracy. The second project adopted a more sophisticated CNN and model balancing.

7 Conclusion

7.1 Summary

The goal of this thesis was to create a tool for the categorization, and the visualization of scraped pages, along with an option to analyse the pages in more detail. To our knowledge the web application created as part of the thesis is the only tool publicly available to perform all of the above listed requirements.

We first analysed the scraped pages stored in Elasticsearch in Chapter 2. In light of the size of the data set at hand we adopted supervised ML for automated categorization. Therefore, a labeled learning data set was built, and improved iteratively. We constructed a CNN and trained a classification model on the learning data set of each iteration. We discovered commonly pre-trained embeddings do not improve the learning results. We compared the learning results on the individual data sets in Section 6.2. The model achieved a prediction accuracy of 88.2%. This accuracy was compared to accuracies achieved by models trained for similar tasks. We concluded our accuracy to be satisfying despite being worse.

We assumed a web graph to be an appropriate way to represent the pages. However, the data set size proved also problematic in the visualization of the web graph as mentioned in Chapter 4. We observed the structure of the web graph to be suitable for community detection. We then considered the LA and the LeA algorithm and applied both. The LA detected more of total and decomposable communities. On the other hand, some of the sub-communities detected by this algorithm were disconnected. The LeA did not produce any disconnected sub-communities. Additionaly, the LeA performed significantly faster than LA as was described in Section 6.1.

A BE with a Rest API was created. The BE categorized the pages via the classification model. The clustering of the pages took also place in the BE. We also created a FE for the displaying of the web graph and page details.

We considered different . The problems arising with the visualization of such a data set and the possible solutions of visualizing such

a data set was discussed in Chapter 4. We adopted the LeA and the LA for the community detection.

To our knowledge, no application for the classification, visualization a web graph

7.2 Future work

7.2.1 Categorization

Presently our classification model predicts categories with 88.2% accuracy. Creating a labeled learning data set with a better quality content or more pages may improve the accuracy.

Currently the classification supports only the English language. A portion of the data set content is therefore categorized incorrectly or as category *Other*. Supporting other languages may improve the classification performance.

7.2.2 User interface

At the current state the category colours are not adjustable. Adding a colour picker would provide the user the option to customize the colours to their liking.

The selection of the clustering algorithm is also not changeable on the UI level. The introduction of a switch for the swapping between the LA or the LeA would present another improvement.

7.2.3 API

The BE provides the option to choose between the LA and the LeA for clustering. However, the API does not mirror this option. The adjusting of the API for this purpose would prove useful.

Page or community details are currently returned in JSON format. We think supporting other formats, such as CSV, would enhance the application.

Bibliography

[1] Introduction to the i2p network [online]. https://geti2p.net/en/about/intro. [cit. 2020-23-01].

[2] Garlic routing and "garlic" terminology [online]. https://geti2p.net/en/docs/how/garlic-routing, 2014. [cit. 2020-05-03].

[3] Dan Abramov and the Redux documentation authors. Redux - a predictable state container for javascript apps [online]. https://redux.js.org/introduction/getting-started/, 2019. [cit. 2019-10-09].

[4] Wesam Al Nabki, Eduardo Fidalgo, Enrique Alegre, and Ivan Paz. Classifying illegal activities on tor network based on web textual contents. pages 35–43, 01 2017.

[5] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. IEEE, 2017.

[6] An alytics Vidhya. Max-pooling visualized [online]. https://www.analyticsvidhya.com/blog/2017/05/25-must-know-terms-concepts-for-beginners-in-deep-learning/pooling/, 2020. [cit. 2020-03-04].

[7] Marcos Assuncao, Rodrigo Calheiros, Silvia Bianchi, Marco Netto, and Rajkumar Buyya. Big data computing and clouds: Trends and future directions. *Journal of Parallel and Distributed Computing*, 75, 01 2014.

[8] J. Bartlett. *The Dark Net*. Random House, 2014.

[9] Michael K. Bergman. White paper: The deep web: Surfacing hidden value. *Journal of Electronic Publishing*, 7, 08 2001. [cit. 2020-06-03].

[10] Vincent Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics Theory and Experiment*, 2008, 04 2008. [cit. 2019-08-16].

[11] Mike Bostock. D3.js - for manipulating documents based on data [online]. `https://d3js.org/`, 2019. [cit. 2019-10-09].

[12] Elasticsearch B.V. Centralize, transform & stash your data [online]. `https://www.elastic.co/logstash`, 2020. [cit. 2020-03-03].

[13] Elasticsearch B.V. What is elasticsearch? [online]. `https://www.elastic.co/what-is/elasticsearch`, 2020. [cit. 2020-01-03].

[14] Elasticsearch B.V. Your window into the elastic stack [online]. `https://www.elastic.co/kibana`, 2020. [cit. 2020-03-03].

[15] Daniel Caldas. React-d3-graph - interactive and configurable graphs with react and d3 effortlessly [online]. `https://goodguydaniel.com/react-d3-graph/docs/`, 2019. [cit. 2019-10-09].

[16] Keunwoo Choi, György Fazekas, Kyunghyun Cho, and Mark Sandler. A tutorial on deep learning for music information retrieval. 09 2017.

[17] Aaron Clauset, M. E. J. Newman, and Cristopher Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70:066111, Dec 2004. [cit. 2019-08-28].

[18] Pandas community. Pandas - data analysis and manipulation tool [online]. `https://pandas.pydata.org/`. [cit. 2020-15-04].

[19] BrightPlanet® Corporation. How large is the deep web? [online]. `https://brightplanet.com/2012/06/04/deep-web-a-primer/`, 2019. [cit. 2020-05-03].

[20] criteo engineering. Web graph by criteo [online]. `http://engineering.criteolabs.com/2014/05/the-web-graph-as-seen-by-criteo.html/`, 2014. [cit. 2019-08-16].

[21] Peter Dayan, Maneesh Sahani, and Grégoire Deback. Unsupervised learning. *The MIT encyclopedia of the cognitive sciences*, pages 857–859, 1999.

[22] Maurice de Kunder. The size of the world wide web [online]. `https://www.worldwidewebsize.com/`, 2020. [cit. 2020-05-03].

[23] NetworkX developers. Networkx - software for complex networks [online]. `https://networkx.github.io/`, 2019. [cit. 2019-09-09].

[24] NumPy developers. Numpy - a package for scientific computing with python [online]. `https://www.nltk.org/`, 2020. [cit. 2020-15-04].

[25] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, page 21, USA, 2004. USENIX Association. [cit. 2020-04-03].

[26] Open Doors. Open doors - persecuted christians [online]. `https://www.opendoorsuk.org/persecution/countries/`, 2020. [cit. 2020-22-04].

[27] Emurasoft. Emeditor - text editor for windows [online]. `https://www.emeditor.com/`, 2020. [cit. 2020-16-04].

[28] David B. Fogel. The evolution of intelligent decision making in gaming. *Cybernetics and Systems*, 22(2):223–236, 1991.

[29] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):4–8, Feb 2010. [cit. 2019-08-16].

[30] Django Software Foundation and individual contributors. Django - a high-level python web framework [online]. `https://www.djangoproject.com/`, 2019. [cit. 2019-09-09].

[31] Python Software Foundation. Cylouvain - a fast implementation of the louvain algorithm [online]. `https://pypi.org/project/cylouvain/`, 2019. [cit. 2019-09-09].

[32] Python Software Foundation. Pickle - python object serialization [online]. https://docs.python.org/3/library/pickle.html, 2019. [cit. 2019-09-09].

[33] Python Software Foundation. Python - an interpreted, high-level, general-purpose programming language [online]. https://www.python.org/about/, 2019. [cit. 2019-09-09].

[34] Python Software Foundation. Shelve - python object persistence [online]. https://docs.python.org/3/library/shelve.html, 2020. [cit. 2020-20-04].

[35] J. George. *Brain Sleep Memory Productivity*. Prowess Publishing, India, 2018.

[36] Shalini Ghosh, Phillip Porras, Vinod Yegneswaran, Ken Nitz, and Ariyam Das. Atol: A framework for automated analysis and categorization of the darkweb ecosystem. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[37] Jean-Loup Guillaume and Matthieu Latapy. The Web Graph: an Overview. In *Actes d'ALGOTEL'02 (Quatrièmes Rencontres Francophones sur les aspects Algorithmiques des Télécommunications)*, Mèze, France, 2002. [cit. 2019-08-16].

[38] Hossam H. Sultan, Nancy Salem, and Walid Al-Atabany. Multi-classification of brain tumor images using deep neural network. *IEEE Access*, PP:1–1, 05 2019.

[39] Kazuyuki Hara, Daisuke Saito, and Hayaru Shouno. Analysis of function of rectified linear unit used in deep learning. pages 1–8, 07 2015.

[40] Alex Hocking, James E Geach, Yi Sun, and Neil Davey. An automatic taxonomy of galaxy morphology using unsupervised machine learning. *Monthly Notices of the Royal Astronomical Society*, 473(1):1108–1129, 2018.

[41] Ratan Hudda, Clint Kelly, Garrett Long, Jun Luo, Atul Pandit, Dave Phillips, Lubab Sheet, and Ikhlaq Sidhu. Self driving cars.

College of Engineering University of California, Berkeley, Berkeley: College of Engineering University of California, 2013.

[42] Facebook Inc. React.js - a javascript library for building user interfaces [online]. `https://reactjs.org/`, 2019. [cit. 2019-10-09].

[43] Stack Exchange Inc. Most popular frameworks according to stack overflow [online]. `https://insights.stackoverflow.com/survey/2018#technology-_-frameworks-libraries-and-tools/`, 2020. [cit. 2020-06-01].

[44] IndoML. A convolusion operation visualized [online]. `https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/`, 2020. [cit. 2020-03-04].

[45] Ankit Jain, Armando Fandango, and Amita Kapoor. *TensorFlow Machine Learning Projects: Build 13 Real-World Projects with Advanced Numerical Computations Using the Python Ecosystem*. Packt Publishing, 2018.

[46] M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

[47] Niels Justesen, Philip Bontrager, Julian Togelius, and Sebastian Risi. Deep learning for video game playing. *IEEE Transactions on Games*, 2019.

[48] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[49] Bekir Karlik and A Vehbi Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.

[50] keras.io/. Keras: The python deep learning library [online]. `https://keras.io/`. [cit. 2020-15-04].

[51] keras.io. A cnn structure by keras [online]. https://blog.keras.io/using-pre-trained-word-embeddings-in-a-keras-model.html, 2017. [cit. 2020-17-04].

[52] Antonin Klopp-Tosser. The open pull request of the adjustment of arrows [online]. https://github.com/danielcaldas/react-d3-graph/pull/271, 2019. [cit. 2019-15-12].

[53] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International conference on machine learning*, pages 957–966, 2015.

[54] Redis Labs. Redis - a in-memory data structure store [online]. https://redis.io/, 2019. [cit. 2019-09-09].

[55] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[56] Wenye Li and Dale Schuurmans. Modular community detection in networks. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, IJCAI'11, pages 1366–1371. AAAI Press, 2011. [cit. 2019-08-28].

[57] Springer Nature Limited. Visualization of how the leiden algorithm detects communities [online]. https://www.nature.com/articles/s41598-019-41695-z/figures/3, 2020. [cit. 2020-04-01].

[58] Encode OSS Ltd. Django rest framework - a flexible toolkit for building web apis [online]. https://www.django-rest-framework.org/, 2019. [cit. 2019-09-09].

[59] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Advances in neural information processing systems*, pages 4898–4906, 2016.

[60] Stephanie McMains and Sabine Kastner. Interactions of top-down and bottom-up mechanisms in human visual cortex. *Journal of Neuroscience*, 31(2):587–597, 2011.

[61] Microsoft. Typescript - a typed superset of javascript [online]. `https://www.typescriptlang.org/`, 2019. [cit. 2019-10-09].

[62] Mozilla and individual contributors. Javascript - most well-known as the scripting language for web pages [online]. `https://developer.mozilla.org/en-US/docs/Web/JavaScript`, 2019. [cit. 2019-10-09].

[63] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006. [cit. 2019-08-16].

[64] Thuy TT Nguyen and Grenville Armitage. Clustering to assist supervised machine learning for real-time ip traffic classification. In *2008 IEEE International Conference on Communications*, pages 5857–5862. IEEE, 2008.

[65] Juraj Noge. Categorizing the dark web [to appear]. 2020.

[66] Visual Paradigm. Visual paradigm online diagrams [online]. `https://online.visual-paradigm.com/`, 2020. [cit. 2020-20-04].

[67] Jeffrey Pennington. Glove: Global vectors for word representation [online]. `https://nlp.stanford.edu/projects/glove/`, 2014. [cit. 2020-16-04].

[68] Pascal Pons and Matthieu Latapy. Computing communities in large networks using random walks. *Journal of Graph Algorithms and Applications*, 10(2):191–218, 2006. [cit. 2019-08-28].

[69] Ahmia Project. Ahmia — search hidden services [online]. `https://ahmia.fi/about/`. [cit. 2020-23-04].

[70] NLTK Project. Natural language toolkit - for the work with human language data. [online]. `https://www.nltk.org/`, 2020. [cit. 2020-15-04].

[71] The Tor Project. Introduction to the tor network [online]. https://www.torproject.org/about/history/. [cit. 2020-23-01].

[72] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009.

[73] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959.

[74] StatCounter. Browser market share [online]. https://gs.statcounter.com/browser-market-share, 2020. [cit. 2020-04-03].

[75] the core Less team. Less - a little more than css [online]. http://lesscss.org/. [cit. 2019-12-09].

[76] V.A. Traag. An implementation of the leiden algorithm exposed to python [online]. https://github.com/vtraag/leidenalg, 2016. [cit. 2020-14-04].

[77] Vincent A. Traag, Ludo Waltman, and Nees Jan van Eck. From louvain to leiden: guaranteeing well-connected communities. In *Scientific Reports*, 2018. [cit. 2019-08-16].

[78] Stanford University. The output of scaling and biases of a neuron. https://cs224d.stanford.edu/lecture_notes/LectureNotes3.pdf. [cit. 2020-07-04].

[79] Ken Wakita and Toshiyuki Tsurumi. Finding community structure in mega-scale social networks. *CoRR*, abs/cs/0702048, 2007. [cit. 2019-08-28].

[80] David F Williamson, Robert A Parker, and Juliette S Kendrick. The box plot: a simple visual method to interpret data. *Annals of internal medicine*, 110(11):916–921, 1989.

[81] Xindong Wu, Xingquan Zhu, Gong-Qing Wu, and Wei Ding. Data mining with big data. *IEEE transactions on knowledge and data engineering*, 26(1):97–107, 2013.

[82] Carlo Zapponi. Active repositories per language on github [online]. https://githut.info/, 2014. [cit. 2019-10-09].

A Abbreviations

We enclose a list of abbrevations with the meaning they represent.

ES Elasticsearch

Tor The Onion Router

I2P The Invisible Internet Project

ORing onion routing

SSK symmetric session key

B Data attachment

The attachments enclosed in the thesis repository[1] contains three source codes with and dumps. The files have the following structure.

`Categorization` contains the source code of the sample data set acquisition and the classification of the manually labeled learning set. The labeled learning set is also included.

`Dark-web-categorization-BE` encompasses the source code of the BE application with the API. The *Redis* and *shelve* dumps are also present.

`Dark-web-categorization` holds the source code to the FE application.

1. https://is.muni.cz/auth/th/noh49/

C Train a classification model

In order to to train a classification model, the following prerequisites need to be installed.

Python 3

Pip 3

- `apt-get install python3-pip`

We now describe the steps to run the classification process itself.

- Download the `Categorization.zip` file and unpack it.
- `cd Categorization`
- `pip3 install -r requirements.txt`
- `jupyter-notebook`
- Open one of the listed url addresses in a browser.
- In the browser, open `keras_classification_model` in the `training` directory.
- From the header menu select *Cell* > *Run all*
- After the computation is finished the following files are dumped into the `models` directory.
  - `DarkWebCategoryModel.h5`
  - `DarkWebCategoryModel.json`
  - `tokenizer.pickle`

  The output of the last cell contains the achieved accuracy, a confusion matrix and the index-category lookup table. The index-category lookup table must also be stored if the newly trained model is to be used.

D Run the web application

D.1 Prerequisites

In order to run the web application the following prerequisites need to be installed and fulfilled.

Python 3

Pip 3

- `apt-get install python3-pip`

Redis

- `sudo apt-get install redis-server`
- (optional) `sudo systemctl enable redis-server.service` (launches *Redis* on OS launch)

Npm 6.4.1 and Node.js 10.14.1 as we cannot guarantee our application to function correctly with different versions of Npm and Node.js

- `sudo apt install node`
- `sudo apt install npm 6.4.1`
- `npm install -g n`
- `node n 10.14.1`

D.2 BE

We describe how to run the BE application on Ubuntu systems. The prerequisites are listed in Appendix D.1.

- Download the `Dark-web-categorization-BE.zip` file and unpack it.
- `cd Dark-web-categorization-BE`
- `pip3 install -r requirements.txt`

- `redis-server` This command needs to be executed from the root directory `Dark-web-categorization-BE`.
- `python manage.py runserver 0.0.0.0:8000`

To use new dumped classification files move them into the `models` directory located in the `Dark-web-categorization-BE` root directory. The index-category lookup table needs to replace the table in `Dark-web-categorization-BE/api/classification.py`.

D.3 FE

We describe how to run the FE application on Ubuntu systems.

- Download the `Dark-web-categorization.zip` file and unpack it.
- `cd Dark-web-categorization`
- `npm install -no-optional`
- `npm run start`

8 Classification results

We detail the statistical values of the accuracy results of the trained models described in Section 6.2.

Plot values	Upper whisker	3rd quartile	Median	1st quartile	Lower whisker	Mean
C1	0.00	0.00	0.00	0.00	0.00	0.00
C2	72.41	70.83	59.74	46.67	40.00	57.59
C3	100.00	76.92	70.84	58.33	10.00	66.43
C4	33.33	0.00	0.00	0.00	0.00	5.00
C5	91.67	61.54	51.92	33.33	22.22	50.66
C6	97.30	95.89	93.46	90.71	65.10	90.70
C7	100.00	66.67	58.34	20.00	0.00	49.00
C8	50.00	50.00	39.28	0.00	0.00	27.86
C9	88.76	83.75	78.33	73.49	53.01	75.52
C10	74.29	70.00	67.82	58.06	33.33	62.82
C11	92.26	90.73	88.54	85.21	80.00	87.84
C12	97.21	96.30	95.10	92.99	91.75	94.69
C13	47.06	34.78	34.05	30.43	21.05	33.58
C14	93.97	91.38	86.27	85.34	81.73	87.54

Table 8.1: The statistical values of training on the first samples. The visualization is shown in Figure 6.1.

Plot values	Upper whisker	3rd quartile	Median	1st quartile	Lower whisker	Mean
C2	97.31	94.97	94.47	93.20	90.40	94.09
C6	96.60	95.97	92.59	91.45	68.57	91.00
C7	98.08	97.87	97.47	91.67	89.58	95.12
C9	98.80	89.74	82.57	78.31	71.05	84.07
C15	86.84	66.67	63.82	43.33	30.56	60.13
C10	76.67	71.43	65.97	52.94	45.16	63.08
C11	94.34	90.00	86.94	82.24	54.04	84.00
C12	69.81	68.18	66.37	60.71	57.14	64.72
C13	65.22	42.86	29.48	17.24	2.86	30.29
C14	95.33	89.08	87.77	84.82	80.19	87.63

Table 8.2: The statistical values of training on the enhanced samples. The visualization is portrayed in Figure 6.2.

Plot values	Upper whisker	3rd quartile	Median	1st quartile	Lower whisker	Mean
C6	95.77	94.66	90.47	90.00	86.92	91.38
C7	100.00	96.00	95.29	92.68	75.56	93.26
C8	92.86	80.68	73.37	64.04	46.67	72.02
C9	88.89	84.21	75.12	65.38	46.43	73.05
C10	77.27	70.97	68.20	65.00	58.82	68.18
C11	92.31	88.03	86.18	85.44	78.12	86.19
C16	98.36	96.86	96.63	95.80	94.44	96.50
C13	62.07	40.74	35.71	30.00	13.79	36.49
C14	94.12	92.44	88.69	84.96	83.61	88.63

Table 8.3: The statistical accuracy values of training on the final samples. The portrayal is depicted in Figure 6.3.

Plot values	Upper whisker	3rd quartile	Median	1st quartile	Lower whisker	Mean
C6	97.97	96.00	94.13	92.09	86.15	93.70
C7	98.15	97.44	96.11	94.87	90.48	95.72
C8	98.63	85.71	75.45	57.97	42.31	72.44
C9	88.89	83.33	64.87	50.00	8.33	60.95
C10	82.35	75.00	66.89	58.62	32.00	64.49
C11	90.45	88.16	86.01	81.37	27.11	80.02
C16	98.71	97.24	96.74	96.54	92.53	96.36
C13	68.42	58.06	45.86	37.04	0.00	44.17
C14	93.75	89.86	86.76	84.68	75.86	86.74

Table 8.4: The statistical accuracy values of training on the final samples with pretrained embeddings. The visualization is shown in Figure 6.4.

Plot values	Upper whisker	3rd quartile	Median	1st quartile	Lower whisker	Mean
first samples	86.11	85.38	84.77	82.22	76.98	83.65
enhanced samples	86.69	85.76	85.01	81.25	77.78	83.81
final samples	88.19	87.73	86.86	85.88	81.37	86.41
final samples E	89.24	88.54	86.11	84.95	72.69	85.41

Table 8.5: The statistical values of the overall accuracy of training on individual data sets. This is depicted in Figure 6.5.

9 Clustering comparison

9.1 Number of communities comparison

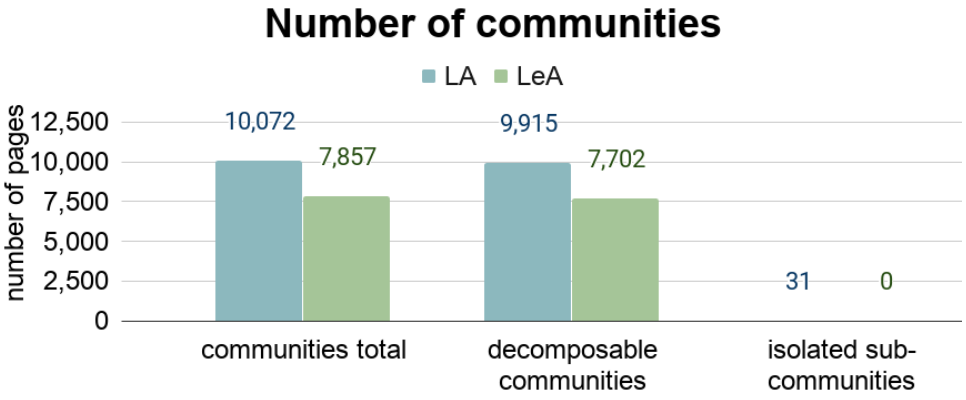

Figure 9.1: The number of communities detected by the LA and LeA compared.

9.2 Partition execution times comparison

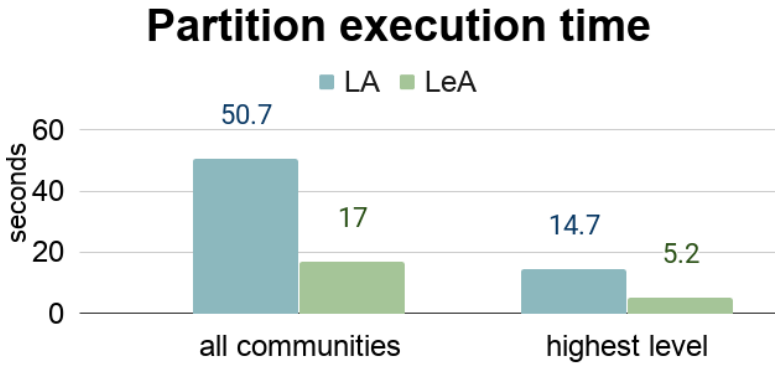

Figure 9.2: The amount of time it took the LA amd LeA to detect communities compared.

10 Imeplementation details

10.1 Classification

10.1.1 Embedding vector

We enclose a portion of an embedding vector for the words *email* copied from the embedding index of the embedding layer.

```
   array([
-0.57792, -0.16606, -0.17226, -0.87941, 0.30588,
...,
0.052887, 0.077703, -0.10182, -0.33831, 0.55555
], dtype=float32)
```

10.1.2 Vocabulary initialization

We detail the initialization of the internal vocabulary. This step precedes the random division of the learning data into training and testing data described in Subsection 5.1.3.

Each word w of each entry in `texts` is represented by an integer n_w. An example of a portion of such a vocabulary is shown in Appendix 10.1.3. Each entry of `texts` therefore corresponds to a vector of integers. All vectors are trimmed or padded to the same length of 1,000 integers.

10.1.3 Data set vocabulary

We enclose a portion of the vocabulary created from our data set.

```
    {
 "1": "porn", "2": "porno", "3": "the", "4": "to", ...,
...,
..., "129988": "3e3zx2drxj", "129989":"tapasoth"
}
```

10.2 API

10.2.1 Classification

The class Categorizer first loads the tokenizer and the trained model. Then the model is compiled with the same attributes as described in Subsection 5.1.3. Both the tokenizer and the model are utilized in the method `categorize`. This method expects the page content as parameter. It applies the tokenizer to the received content. The output is padded or concatenated to the same length of 1,000 digits. The model predicts the category alias of the modified content. Lastly, the alias is converted to the name of the category.

10.2.2 Models

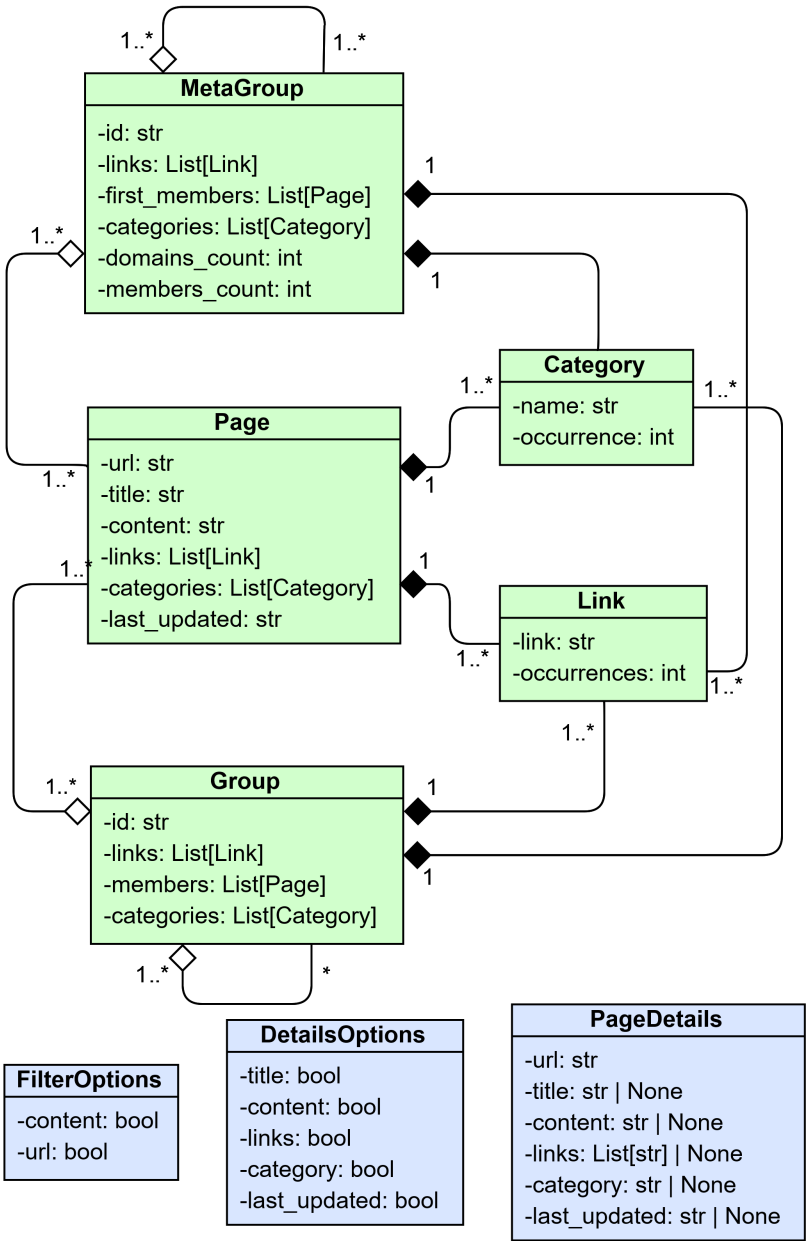

Figure 10.1: A class diagram of the classes used in the BE. The diagram was created leveraging Visual Paradigm [66]

10.2.3 Response example

We now detail an example of handeling a response. The API structure is described in Subsection 5.3.2. Let us assume the API received the request

```
GET /api/pages/bylink/?content-type=json&id=2.0
.
```

The client now expects the API to return sub-communities of the community with the id *2.0*. The ViewSet `GroupsByLinkViewSet` is assigned to the used endpoint. The method used in the request is *GET*. Therefore the *list* function of `GroupsByLinkViewSet` is called. The url provided contains the `id` parameter. The helper method `get_subgroups_of_group` with the provided `id` is called.

The method caches and returns a list of `Group` objects. However, this response would be too sizable for the server-client communication. That is why the `Group` objects are converted to `Meta_Group` objects. A `Meta_Group` object doe not contain all of the community pages. It contains only the first 10 pages as `Page` objects. `Meta_Group` also contains an attribute representing the number of the pages. The list of `Meta_Group` objects is serialized via the `MetaGroupSerializer`.

At last a `Response` object is constructed with the serialized result as its data. The response is returned to the client.

10.2.4 Page acquisition

We detail the implementation of the different page acquisition methods.

Method `basic_search` is used for the filtering of nodes based on page-content or page-ulr.

Method `fetch_chunk` retrieves portions with pages called *chunks* from the database. In this thesis a chunk contains 500 pages. Each response from the database holds a chunk and a *scroll_id*. This id is sent in the next request to the database. The enclosed chunk in the next response from *Elasticsearch* depends on the scroll_id. The search context of *Elasticsearch* is kept open for 1 minute. This is defined by `scroll` attribute. Each request extends this time by another minute.

Method `fetch_all` gets all pages from the database. In this method `fetch_chunk` is called repeatedly until the response carries

no pages. Each chunk is converted to a dictionary of `Page` objects with the page url as key and the whole page as value. Also, each batch is stored on the disc in a *shelf*. The keys in the *shelf* are indexes of the batches and the values are the whole batches. After all pages from the database are obtained the content is removed from the pages. Also, invalid links are deleted from the pages. Then the pages are cached in *Redis*.

10.3 FE

10.3.1 The remaining FE file structure

`node_modules` contains imported libraries including *React.js*, *Redux* or *d3*.

`public` encloses a *.ico* file[1] and a html file which is the default entry point when the application is started.

`commont.d.ts` holds types used heavily across the application, e.g. *Action*. Types in this file are available without importing them to all files in the project.

10.3.2 State modification structure explained

The most basic files which only include string constants are situated in the folders named `actionTypes`. These are utilised as action types in actions. An action is a simple objects containing a type and an optional payload. Actions themselves are returned by action creators (AC).

ACs are functions returning actions and can be found in folders called `actions`. They can be as simple as those present in the file `nodesActionCreators.ts`. But they can be more complicated such as the ACs `fetchNodes.ts` and dispatch multiple simple ACs. The AC `fetchNodes` is described in more detail in Appendix 10.3.3. The purpose of an AC is to be injected into reducers, i.e. dispatch `actions`.

---

1. A picture with the dimensions 16x16 pixels used by the browser to represent the web page or application. It is usually displayed in the tab in which the application is opened.

The dispatching of actions enables the changing of the state via reducers situated in the `reducers` folders. The state is a single immutabe[2] object and is used in the whole application. A reducer is a pure function[3] receiving the current state and the dispatched action as its arguments. It then returns the newly computed state. A reducer builds a new state only if the type of the given action is recognized. If not, the previous state is returned unmodified. The state object received or returned by the reducer does not need to be the entire state (app-state). A reducer may be responsible for only a part of the app-state. However, the root reducer is responsible for the whole app-state.

10.3.3 Detailed fetchNode AC

The AC `fetchNodes` and the folder it is placed in share the same name. For easier testing purposes the main logic of this AC is put into a function which receives the simple ACs as dependencies. When this AC is called it first dispatches a simple AC to indicate the fetching has begun. After that an identificator (id) is initialized. This id is later used to create an error object in case of failure. Next, the fetching itself begins. The fetching in `fetchNodes` is realized with the library *isomorphic-fetch*. The fetch function of this library expects the first argument to be the url address of the resource. The second argument is an object describing further details of the request and is optional. Such an object may contain the request method, headers or the payload. If the request does not result in error the response status is checked. After the fetching is complete a success AC with the acquired response is dispatched. If an error is caught during the fetching a failure AC is dispatched. The payload of this AC is an error-object with the id and error message if any.

---

2. The object cannot be adjusted directly. Instead, a new modified object is returned and the original one stays unmodified.
3. The return value of a pure function is only dependent on its input values. A pure function has no side effects.