Masaryk University
Faculty of Informatics

The Categorization of the Darkweb

Master's Thesis

Bc. Linda Hansliková

Brno, Spring 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 advice and patience.

Abstract

The future Categorization of the Dark Web abstract

Keywords

Web Application, Dark-web, Python, RestApi, Django REST, TypeScript, React, Redux, d3 Graph

Contents

List of Figures

1 Introduction

2 Data set analysis

The data set we were working with consisted of pages which had been scraped[1] from the dark web and stored in Elasticsearch (ES) [12]. This chapter will describe the clear web, deep web, and the dark web, the Elasticsearch database, and the structure of the stored pages.

The Internet consists of networks from all over the world. The networks are connected and together they comprise a global network. The Internet can be divided into the clear web 2.1, and the deep web 2.2 [8].

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 [65]. 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 [21].

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 of the fact the content is not indexed. The size was estimated to be 4,000 to 5,000 times larger than the clear web [18].

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

1. Scraping is a method used for collecting data from web pages with automated software rather than doing so manually.

2.2.1 Dark web

The dark web provides anonymized services and user identities. It is therefore leveraged for activities like accessing or publishing illegal or censored material, e.g. the Bible, whistle-blower secrets, or child porn. It is also used for trading 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 [7]. The onion router (Tor) 2.2.2 or the Invisible Internet Project (I2P) 2.2.3 are two of the networks comprising the darknet.

2.2.2 Tor

The Tor network [62] is accessible through the Tor browser or Tor proxy. Tor makes use of *onion routing* (ORing) [24]. ORing describes routing where each node, except for the originator and the target node, knows only its predecessor and successor. The originator chooses a list of nodes from which it creates a circuit between itself and the target node. Nodes in this circuit are determining their successor leveraging public-key cryptography. For example, A is the originator and D the target node. A creates a circuit consisting of itself and of nodes B, C and D. A creates a fixed-sized cell with encrypted information about the circuit nodes with a message using their respective public keys. A sends this cell to B. B removes one layer of the cell using its private key in order to get the address of its successor, in this case C. B cannot remove an additional layer to the successor of C because it lacks the private key of C. B sends the cell to C. C removes another layer and sends the cell to D. D removes the final layer and reads the information. Now D is able to communicate with A through D and C with only knowing about itself and D. Determining the original source and target node is therefore difficult. ORing was introduced in the 1990s to ensure privacy.

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* (GRing) [2]. GRing 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*[2]. Each clove contains their 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 browsers provide anonymous access to the clear web as well as to the dark web. Both are open-source and free to use.

2.3 Elasticsearch

The data collected from the dark web was store in Elasticsearch. ES is a distributed, open source search engine [12] and offers a fast full-text search. Another benefit of ES are documentation and supported tools, such as Logstash [11] for processing of data or Kibana [13] 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.

2.4 The data set

The dark web pages were acquired via web scraping. 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 will be described in the following list 2.4. Each description will include a concrete example from the database. Fields beginning with an underscore are assigned to every document implicitly.

---

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

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

_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*.

3 Classification

A part of this thesis was the categorization of the scraped pages. Categorization in this context means the procedure of assigning categories (labels) 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. The 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* [64] 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 [41]. 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] [71] [6]. ML is used widely with Big Data. ML is also used in the language processing field [41]. 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 [63]
3. A data set too vast or complex for traditional or manual data processing.

are examples[36]. Also, ML is used frequently in games for a more natural game experience [26].

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 [43]. 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 [42].

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 [20]. 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 [35].

3.2.3 Supervised learning

SLr is suitable to use for the classification of data. During training SLr relies on labeled data [63]. 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 [58].

3.3 Artificial neural networks

An ANN is a ML system inspired by animal brains [63]. 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 [31]. 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 is comprised of 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 later in subsection (AcF) 3.3.5. Neurons exist in groups called *layers*. Layers used in this thesis were *embedding layers* 3.3.1, *convolutional layers* 3.3.2, *polling layers* 3.3.3, and *dense layers* 3.3.4. 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 is 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 is passed to the first layer. Data is passed from one layer to the next one. Every layer modifies the received data before passing it on. The manner in which the data is modified depends on the type of the layer. After the data is processed by the last layer the output model is evaluated using the testing data. Improving the model

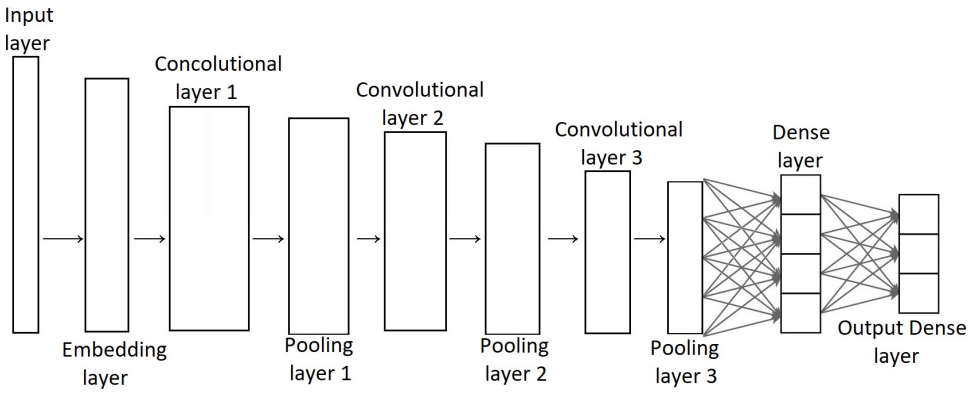

Figure 3.1: An illustration of the CNN implemented in this thesis. The two portrayed dense layers are a modified image from an article on deep learning [15]. The individual layers are detailed in section 3.3.

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 [47] 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 [49]. 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 between them. From the vectors of the previous example the result vector could imply $queen - princess = is\ predecessor\ of$.

The EmbLr may also use a finished 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 [4]. A ConvL decreases the size and complexity of the input by convoluting the input. The convolution of data is 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* [53]. 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 a later subsection activationFunction.

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 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.

---

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.

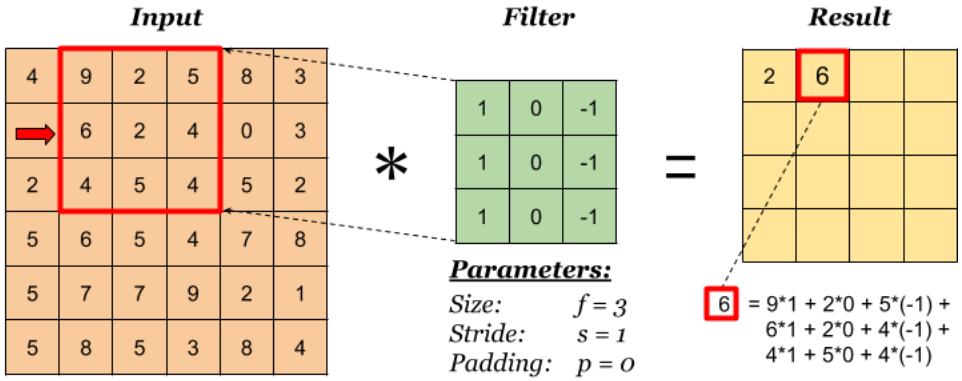

Figure 3.2: The visualized process of a ConvO [39]. The very left square is the input – a 2D image of size 6x6 pixels. The square in the center depicts a 3x3 kernel with no padding. The stride of the kernel is 1 pixel. The very right square represents the partial result after the kernel was applied to two sections of the input. The final output will be of size 4x4 pixels. The ConvO itself is enumerated under the output image.

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 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* [4]. 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 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

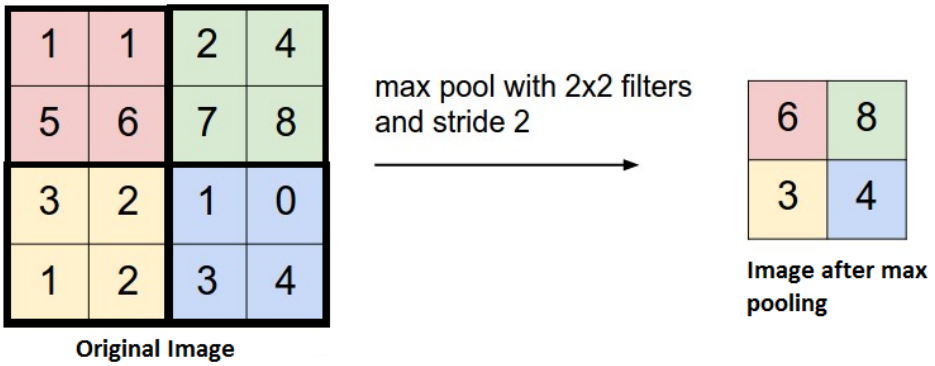

Figure 3.3: The visualized process of max-pooling with filter size of 2x2 pixels and stride length of 2 pixels by Analytics Vidhya [5]. The input is a 2D image and is depicted by a square with scalar values. The size of the image is 4x4 pixels. The square on the right is the result after max-pooling is applied on the input. The output image is of size 2x2 pixels. The max-pooling process is described in subsection 3.3.3.

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 [4]. 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 enu-

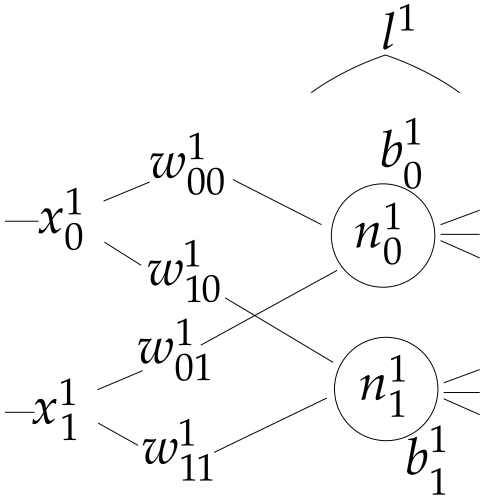

Figure 3.4: The visible portion of this network consists of a DnsL l^1. This layer is composed of two neurons, n_0^1 and n_1^1 and their corresponding biases b_0^1 and b_1^1. Each neuron receives two inputs x_0^1 and x_1^1 with weights w_{00}^1, w_{01}^1, w_{10}^1, and w_{11}^1 respectively. The operation 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 $w_{00}^1 \cdot x_0^1 + w_{01}^1 \cdot x_1^1 + b_0^1$.

merated as a multiplication of two matrices w and x with the formula

$$z_n^l = w_n^l x^l + b_n^l \text{ [69]}.$$

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 activationFunction. 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 activation function (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 [44]. 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 [34]. The function itself is shown in figure 3.5. It is expressed as

$$\sigma(x) = \begin{cases} 0, & \text{for } x \leq 0 \\ x, & \text{for } x > 0 \end{cases} \quad [34]$$

where σ represents ReLU. ReLU is used for its reduction of activation since only positive input values are having a non-zero output. Another advantage is information of positive input values is not lost.

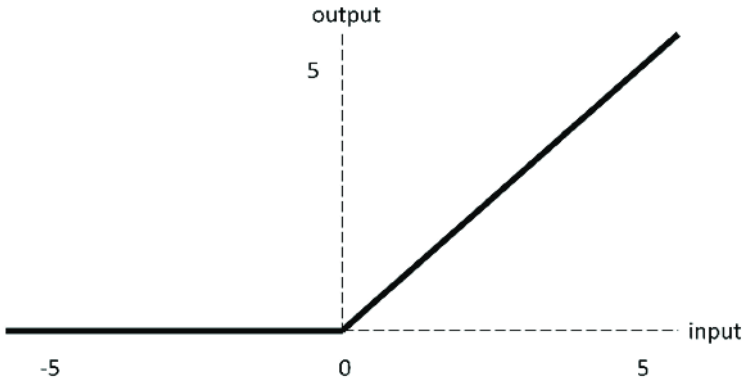

Figure 3.5: The ReLU AcF from the Multi-classification of Brain Tumor Images using Deep Neural Network [33] article.

Softmax is an AcF leveraged for the designation of probabilities of an input belonging to beforehand given labels [63]. 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 [63]. 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

$$LsF(x, y, \hat{y}) = Utility(\text{result of using } y \text{ given an input } x) - Utility(\text{result using } \hat{y} \text{ given an input } x) \text{ [63]}$$

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 [40]. 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

$$-\sum_{c=1}^{M} y_{i,c} log(p_{i,c}) \text{ [40]}$$

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 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) [63].

The weights and filters of individual layers are modified so that after the next forward-propagation[8] the LsF output is smaller. 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 .
8. The flow of the data in the direction from the input layer to the output layer as opposed to backward-propagation.

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 is not efficient in minimizing the output of the LsF. An optimizer is an algorithm leveraged for the optimization of the learning rate.[9] The optimizer is applied during the updating of the weights by the back-prop. The goal is to make the output of the LsF converge in a smaller number of epochs to the global minimum of the LsF. The optimizer used in this thesis is RMSprop. RMSprop updates the learning rate dynamically based on the history of the weight values.

9. The value by which weights are updated in order to minimize the LsF output.

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 is displayed as a web graph [32]. 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 2.4 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 incomprehensible as too

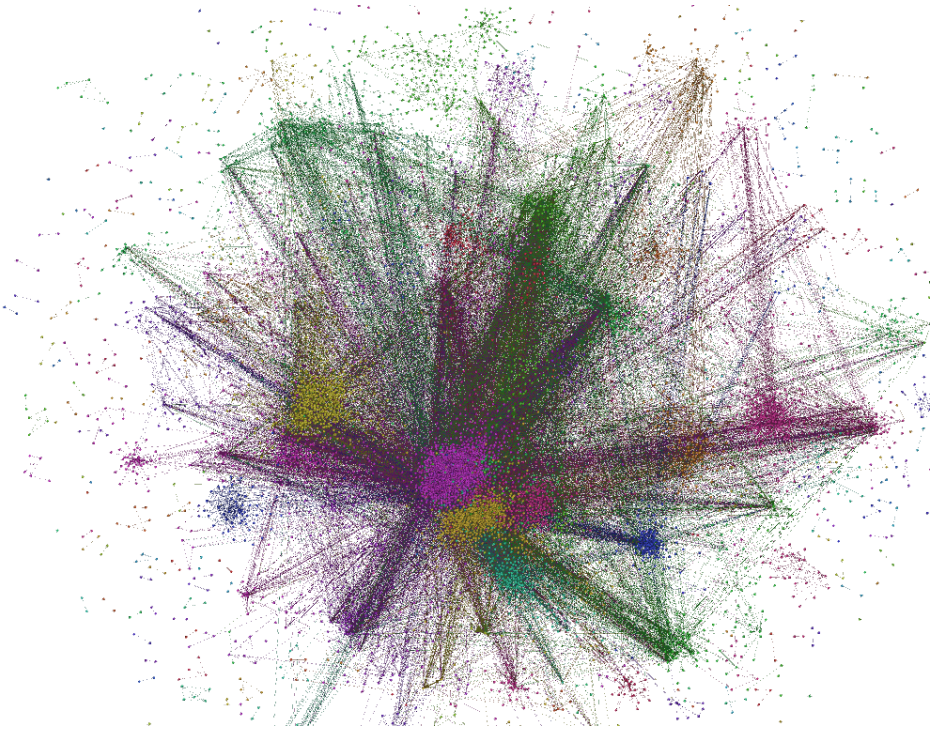

Figure 4.1: Web graph by Citeo made consisting of circa 600 000 domains and 16 billion links. [19].

much visual data worsens the quality of the user's information processing [54]. 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 [27]. 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 [57] 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 [50].

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 [9]. 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.

1. 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.

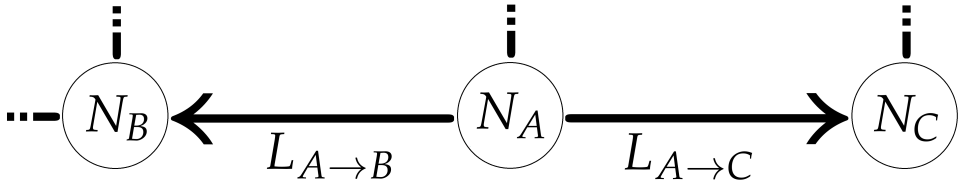

Figure 4.2: The visible portion of the exampled graph depicts nodes N_A, N_B, and node N_C. There are links $L_{A\to B}$ and $L_{A\to C}$ between nodes N_A and N_B, and N_A and N_C respectively. Other links are displayed partially and are connecting the nodes to the rest of the graph.

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

2. 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*.

3. 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.

4. Discard the empty communities.

   - Community C_A is now empty and is therefore discarded.

5. 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.

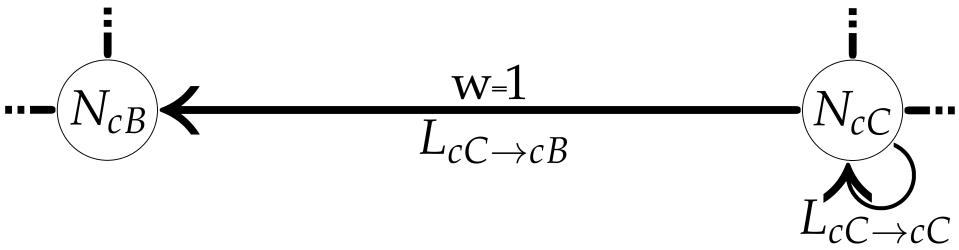

Figure 4.3: This graph is the result of applying the above listed steps to the previously shown portion of the example graph from figure 4.2. The visible portion of the graph contains nodes N_{cB} and N_{cC}, and a link $L_{cC \to cB}$ with a weight of 1 between them. A self-loop $L_{cC \to cC}$ on N_{cC} 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 [9]. Namely the algorithm of Wakita and Tsurumi [70], of Pons and Latapy [60], and of Clauset, Newman and Moore [16]. 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 [68]. 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 [51]. The steps are compared to the steps depicted in the characterization of LA 4.2 and described using the visualization.

a) This phase corresponds to step 1. from the LA description.

   - *Part 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 2., 3., and 4. from the LA description.

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

c) This phase is called **refine**. Steps 1. and 2. are applied to nodes within communities. This results into the further partitioning of communities into sub-communities.

---

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.

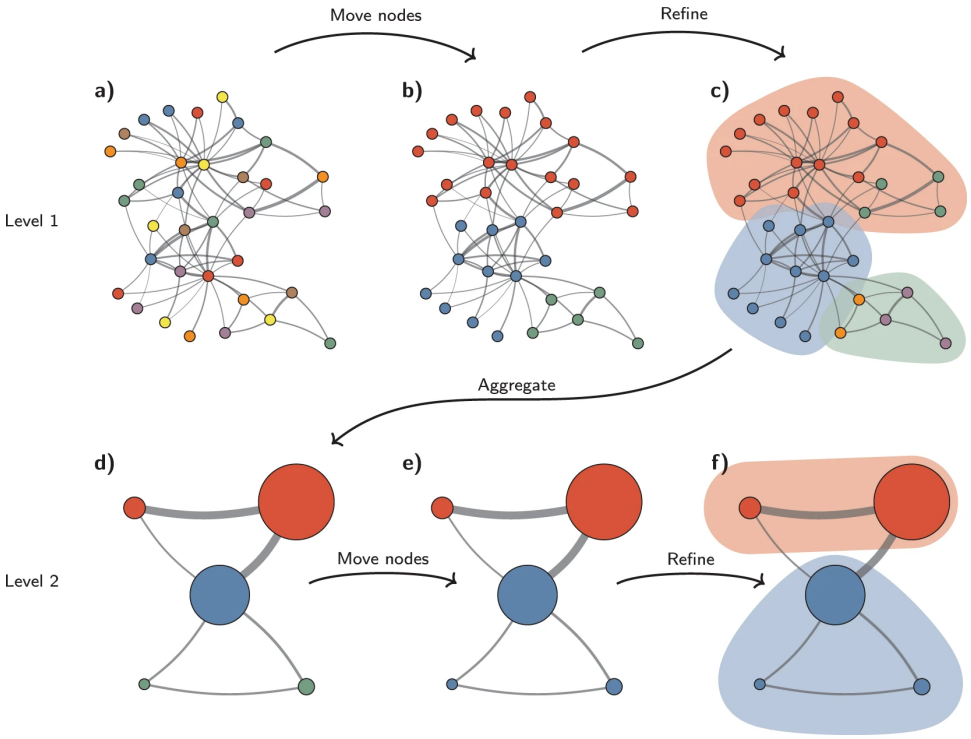

Figure 4.4: The visualization of the principle of LeA [51] by the authors of *From Louvain to Leiden: guaranteeing well-connected communities* [68]. Steps *a)* to *d)* are described in the characterization of the LeA principle 4.3. Steps *e)* and f) show a part of a second iteration of the algorithm. The algorithm ended in step *f)* because no further improvement was achieved by **refining** the communities.

- *Part 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 5. from the LA description 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.

- *Part 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 part *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 his thesis 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 of retrieving all the available information about the pages was demanded.

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

In order to categorize the pages depending on their content a CNN needed to be created. For that a project called `Categorization` was built. For the training of a CNN a suitable training and testing data set needed to be created. The desired output of the CNN was a model able to categorize the pages with a reasonable accuracy. This chapter describes the steps taken to achieve the listed goals and the technologies used.

5.1.1 Technology overview

The project was written in Python. *Python* is a widely used interpreted programming language known for readability and portability [30]. 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 [25] 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) [61]. 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 [23]. 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 [17] 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 [45]. Keras is a library for the implementation of ML in Python. ML is closer characterized in the section 3.2. Keras supports the tokenization of text and the tools needed to create 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 created 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. 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 found. 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 6.1.

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. Gambling 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 found. We will call this data set *final samples*. The 9 categories detected are detailed in listing 5.1.2.

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 1196 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 took place in the file `KerasModelDataset.ipynb`.

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

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

the content the document was considered English or not. The threshold was 20%. Non-English documents were removed from the data frame.

Next the labels had to be preprocessed. The labels were 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 was created. The indexes would be used as category aliases.

Afterwards two empty lists were created, `texts` and `labels`. For each row of the data frame the content was appended to list `texts` and the category was converted to the corresponding alias and appended to the list `labels`. The text is associated with the corresponding 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 was created from all words from `texts`. Each word w of each entry in `texts` is represented by an integer n_w. Each entry of `texts` therefore corresponded to a vector of integers. All vectors were trimmed or padded to the same length of 1,000 integers. The list `texts` corresponded to a matrix. The matrix was stored in a list of matrices called `data`. The list `data` was split into a training and testing set. One fifth of `data`, meaning 3,457 entries, was randomly chosen to be the test set. The rest was the training data.

Each category alias in `labels` was assigned an index i. Each entry of the list was replaced by an OHV[6] . The value 1 was on the ith position of the OHV. The length of the OHV corresponded to the number of categories. The list `labels` became a matrix after this modification, meaning a list of vectors.

The next step was 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 [59]. However, using that matrix changed the learning results only marginally as opposed

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.

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 therefore decided not to use the trained embeddings.

The last step before the training itself was to create the shape of the CNN. The Keras documentation offers a sample of a CNN structure meant for text classification [46]. We used this structure and figure 3.1 depicts the structure composition.

The EmbLr received information about the vocabulary size, the number of words on each pages and the length of the output vectors. The vocabulary was limited to 20,000 most frequently occurring words. The number of words in the content of each page was limited to 1,000. The EmbLr therefore awaited an input of vectors with the length 1,000. The output of this layer would be embedding vectors with the length of 100.

The second layer was a ConvLr with 128 neurons and a kernel with the dimensions 5x5. The activation function of this layer was ReLU described in subsection activationFunction. Next was a PoolLr with filter dimensions of 5x5. The method used in the pooling layer was max-pooling. The following layer was a ConvLr followed by a PoolLr and another ConvLr all equivalent to the first layers of the same types. The seventh layer was 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 followed. The last layer was a dense lauer with the activation function softmax. The number of neurons in this layer was the number of categories.

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

After the training was finished the model was exported. It then was used in the 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. This amount of pages 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 sections 4.2 and 4.3 of chapter respectively.

5.2.1 Technology overview

In the beginning we used the library *NetworkX* [22] for the graph creation. NetworkX is a Python library for the managing of complex networks. The advantages of this library are 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 LA implementation was too slow for the size of our data set. The built in LA didn't succeed to divide about 40,000 pages after 30 minutes. We therefore decided to use a different implementation. The *cylouvain* library *cylouvain* was introduced.

Cylouvain is a faster implementation of LA. The division of 90,275 pages into communities took about 18 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* [67] library was introduced.

Leidenalg is an *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 subcommunities of communities. Leidenalg is built on the *python-igraph* library.

Python-igraph (igraph) another library for the managing of complex networks. The advantage of this library the availability of the before mentioned implementation of the LeA. Another advantage is igraph disposes of an implementation of the LA. This implementation takes about 5 seconds to divide 90,275 pages into communities.

Also, no disconnected sub-communities of communities are created. Therefore this implementation was also leveraged.

The two utilized implementations are compared in the section 6.2.

5.2.2 Implementation

The clustering is implemented in the back-end 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 described in the previous subsection 5.2.1 require the nodes to be in igraph format. To create an igraph from the nodes we first need to convert them into integer aliases. The aliases will constitute the vertices of the graph. The node links also need to be in the form of integer aliases. These link aliases will compose the edges of the graph. Now the graph which was created using the igraph library can be filled with the edges and the vertices. Next, isolates are filtered out and deleted from the graph vertices. Afterwards it is decided whether to use the LA or the LeA implementation. The decision is made based on a boolean flag. Both implementations return 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 5.1. 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.

7. Ids in case of pages are urls. Ids in case of communities are specific strings

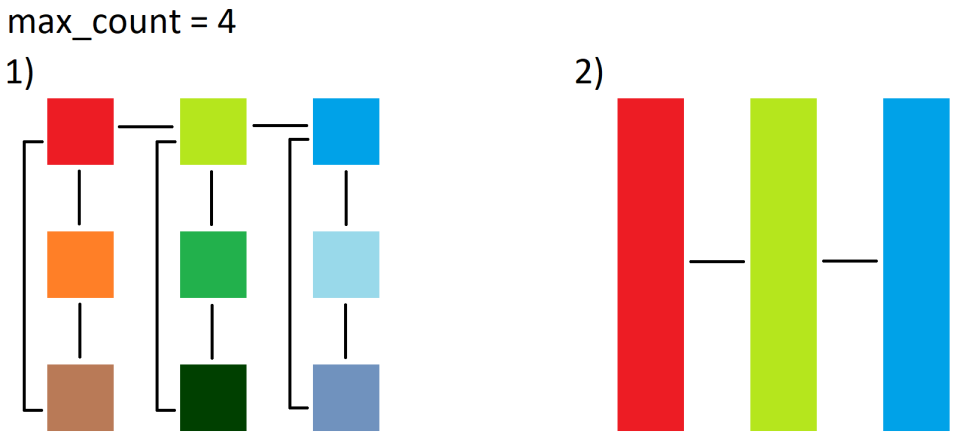

Figure 5.1: An example of multiple cluster cycles. Max count is four. The image on the left represents a hypothetical output of the first cluster cycle. It contains 9 squares each of a different colour with undirectional links between them. Each square depicts a community. The number of the communities 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.

5.3 API

The scraped pages of the dark-web were being stored in *ElasticSearch*. We created a back-end application (BE) in order to perform various operations on the data-set before sending it to the FE. Such operations involve resource intensive processing of large volumes of data, and caching. We decided to create a Python BE. 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 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 [28]. 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 [52]. 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 220,000 pages from the database and circa xxx 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* [48] was used. It 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* and *links*, an object serializer was leveraged along with Redis. We decided not to write our own but to utilise the Python *pickle* module[9] [29] (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 xxx seconds to serialize xxx pages. The deserialization of the same number of pages took xxx seconds.

5.3.2 Implementation

5.4 Front-end

For users to be able to see the data acquired from the BE in a reasonable way a FE application was created. The goal of this 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 category of the pages or communities was to be readable

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

from 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.

After the application is loaded the UI is composed of a *header* with the name of the application *Dark web categorization*, a *loader* and a *sidebar* on the right hand side. The application resembles at this point the image in Figure 5.2.

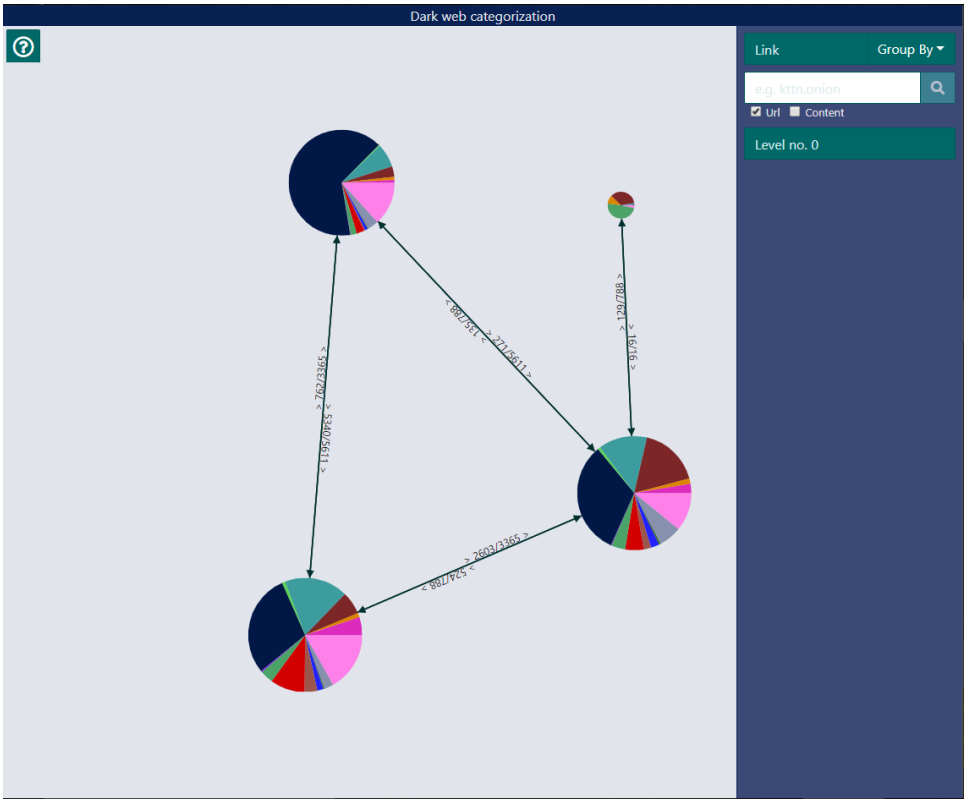

Figure 5.2: The view after the application has loaded.

The *sidebar* contains several *input fields* and *buttons* in a column. At the very top of the *sidebar* a *drop-down button* (DD) is present.

The DD sets the mode according to which the pages are grouped into communities. There are two modes available, *link-mode* and *category-*

mode as can be seen in Figure5.3. If *link-mode* is selected, the pages are divided by the LA4.2 depending on the connections between them only. If *category-mode* is selected, the pages are divided into groups by categories. The pages in the groups are further divided into communities according to links between them, as if in *link-mode*.

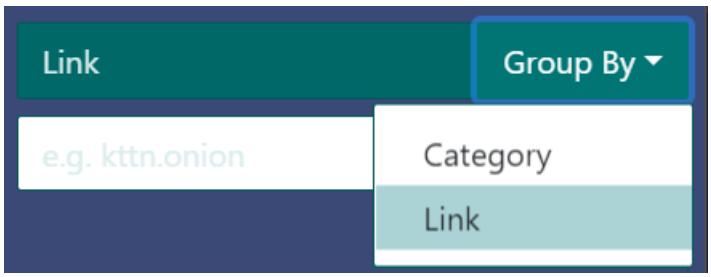

Figure 5.3: The DD for the selection of the grouping-mode.

An *input field* with a *submit button* was placed underneath the DD for the purpose of filtering nodes according to a search phrase as can be seen in Figure5.4.

Figure 5.4: The input field with submit button for filtering.

The last element shown is an indicator of the current level - how many times the user zoomed into a community. The indicator has a *zoom-out button* placed next to it.

The moment the data is retrieved from the BE the *loader* gets replaced for a graph. The graph-nodes represent either the communities the pages are partitioned into or the pages themselves. If the graph contains any isolated nodes (isolates) a *mock-community* is displayed containing all isolates. This community cannot be zoomed into. A community node is depicted as a *pie chart*. The individual sectors of the *pie chart* represent the categories of the pages belonging into the community. This is portrayed in Figure5.5. It was difficult to distinct between a small community of only one category and page

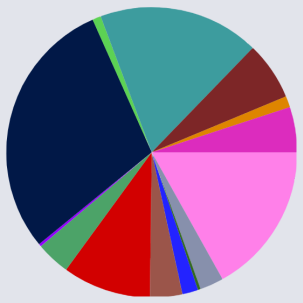

Figure 5.5: The representation of a community node. The individual sectors of the pie-chart illustrate the category-composition in the community.

Figure 5.6: The representation of a page node. The colour of the square symbolizes the category of the page.

nodes. A page node is therefore differentiated from a community node by depicting the node as a square shaped symbol. The colour of the square corresponds to the category of the page. An example of a page is illustrated in Figure5.6. It is possible for a level to depict communities and pages at once. The links between the nodes visualize the links between pages or communities.

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.

Links to other pages displayed as a list of url addresses. There are up to ten links visible in the sidebar. The remaining links, if any, are downloadable in a text file. Figure5.8 contains a sidebar view with page details.

The details of a community consist of grouped information of its members and include the following:

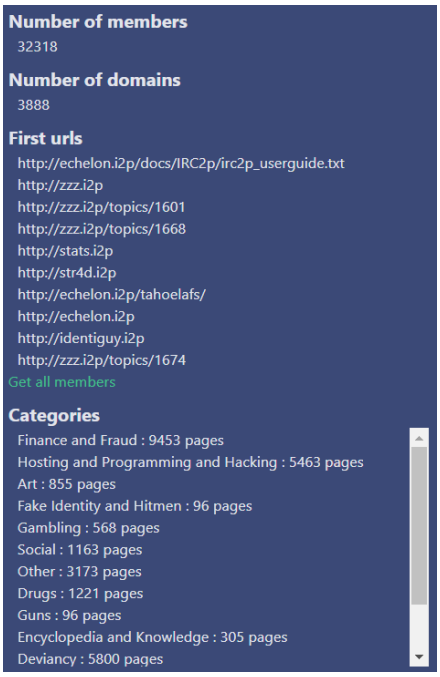

Figure 5.7: The sidebar with details of the selected community.

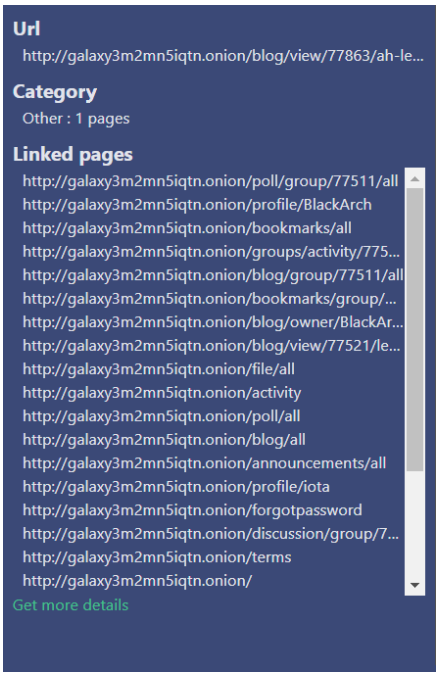

Figure 5.8: The sidebar with details of the selected page.

Category composition which is aggregated from the categories of all the pages of the community. Each category is represented by its name and the percentage of its relevance in the community.

Page url addresses (urls) of members belonging into the community. There are up to ten urls visible in the sidebar. The remaining urls, if any, are downloadable in a text file.

Urls count represents the number of all the pages belonging into the community.

Figure5.8 contains a sidebar view with community details.

In case of the need of further information a link, also present in the *sidebar*, can be clicked. If clicked, a *pop-up window* with detail-options is displayed. Details may include the *title*, *category*, *links* and *page-content*, depending on the user's selection. *Urls* of the pages are

present by default. The window can be seen in Figure5.9. After selecting the needed options and clicking the *download button*, a *text file* with the desired information is downloaded. The page or community-members with their details are depicted in JSON format. This functionality was required for the scenario of the user needing to download all available data about members of a specific community.

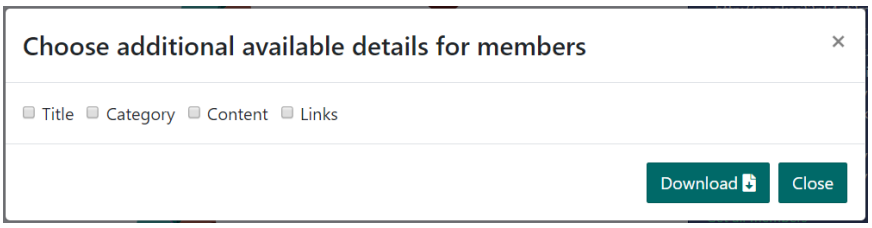

Figure 5.9: The pop-up window with detail-options for selection. These options dictate which details will be included in the downloaded text file.

A graph node representing a community can be double clicked. After doing so, a new graph is shown. The data of this graph consists of the members of the clicked community. We call this process *zooming*. In case the parent community contains too many sub-communities, it is displayed as a single community-node. The zooming-in or -out of communities adjusts the *zoom-level*. This level is represented by a number and is visible below the *node-filter*. If the current zoom level is more than zero, i.e. at least one community-node was double clicked, a button for zooming out appears next to the *level indicator*. It is shown in Figure 5.10.

Figure 5.10: The level indicator with the zoom-out button. After the zoom-out button is clicked, the user is shown the communities of the previous level.

If further zooming is not possible, the user reached the maximum level. Each community may have a different maximum level, depending on the number of its pages and its structure.

5.4.2 Technology overview

JavaScript [56] (JS) is the most favoured programming language used for creating web applications [72]. 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) [55] 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* [37] (React). React is one of the most favoured JS frameworks [38]. 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. However, 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[10]. 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 visualization of the web graph alone was realized using the *react-d3-graph* library [14] (RD3). RD3 is an implementation of the library D3.js [10] made more convenient for the use with React.js.

5.4.3 Implementation

The FE project consists of three folders and several configuration files. The folder `node_modules` contains imported libraries including *React.js*, *Redux* or *d3*. The next folder named `public` encloses a *.ico* file[11]

10. The process of an object being provided with information
11. 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.

and a html file which is the default entry point when the application is started. The last folder `src` contains the source code itself.

As previously mentioned, the FE is written in TS which has the advantage of readability and easy navigation. There are, however, also disadvantages. One of them is the need of a TS file with the types (typing) for every used library. Typings for popular libraries are often downloadable as modules. If a library has no ready-to-download typings own ones need to be written. In our case the typings for the library *react-d3-graph* were custom made. They can be found in the folder `@types/react-d3-graph`. The file `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.

Objects passed between functions also need to be typed. Those models are stored in the folder `models`. 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.

The visual aspect is implemented using Less [66] which is a language extending CSS with improvements such as the possibility of using variables. The Less classes are divided into files depending on the element they are meant to modify. These files were placed into the folder `styles`.

The remaining folders each represent a different part of the UI. The structure of their sub-folders is similar. Therefore it is sufficient to describe them as a whole. Folders named `utils` contain files with helper functions such as converters between server and client models. `Constants` contains folders with string constants or simple functions which return a string depending on the input. The rest of the folders represent some part of the Redux framework.

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). AC are functions returning an action 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 AC `fetchNodes.ts` and dispatch multiple simple ACs. The purpose of an AC is to be injected into reducers, i.e. dispatch them.

`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 created. 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.

The dispatching of actions enables the changing of the state via reducers situated in the `reducers` folders. The state is a single immutabe[12] object and is used in the whole application. A reducer is a pure function[13] receiving the current state and the dispatched action as its arguments. It then returns the newly computed state. A reducer creates 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 just a part of the app-state. However, the root reducer is responsible for the whole app-state.

The folders `components` contain files with React components. They define the skeleton of the UI with the specified behaviour. The folders `containers` hold files with React containers. A container is a file with access to the app-state. It is responsible for passing data to components.

---

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

6 Evaluation

6.1 Classification results

6.2 Clustering results

7 Conclusion

7.1 Evaluation

7.2 Future work

7.2.1 User interface

Implement a colour picker so the user would be able to select colours belonging to categories. Implement a switch for the option of which algorithm to choose for the clustering.

7.2.2 API

7.2.3 Categorization

Improve the model for categorization by for example feeding it more samples of certain categories. Support other languages

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] 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.

[5] 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].

[6] 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.

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

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

[9] 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].

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

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

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

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

[14] 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].

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

[16] 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].

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

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

[19] 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].

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

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

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

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

[24] 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].

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

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

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

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

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

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

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

[32] 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].

[33] 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.

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

[35] 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.

[36] 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.

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

[38] 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].

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

[40] 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.

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

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

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

[44] 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.

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

[46] 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].

[47] 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.

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

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

[50] 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].

[51] 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].

[52] 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].

[53] 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.

[54] 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.

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

[56] 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].

[57] 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].

[58] 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.

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

[60] 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].

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

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

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

[64] 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.

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

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

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

[68] 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].

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

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

[71] 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.

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