Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Detecting Propaganda Articles by its Internet Distribution Pattern

Bc. Ondřej Bouček

Supervisor: Garcia Sebastian, Assist. Prof. PhD
Supervisor–specialist: Babayeva Elnaz
Field of study: Open Informatics
Subfield: Artificial Intelligence
May 2022

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Bouček Ondřej**	Personal ID number:	**474654**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence**		

II. Master’s thesis details

Master’s thesis title in English:

Detecting Propaganda Articles by its Internet Distribution Pattern

Master’s thesis title in Czech:

Detekce propagandistických článků podle šíření na internetu

Guidelines:

The goal of this thesis is to test whether it is possible to detect the distribution of computational propaganda by tracking the spread of an article through the Internet. The student will develop and improve the searching tool developed by Stratosphere Laboratory to find which web pages are linking and referencing an article. Then a graph representation of an article distribution found by the searching tool will be created. Next, it is expected to collect a data set of propaganda and non-propaganda URLs. Lastly, the student shall develop selected machine learning models to test whether it is possible to detect propaganda using the graph representation approach.

Bibliography / sources:

[1] Woolley, Samuel C., and Philip Howard. "Computational propaganda worldwide: Executive summary." (2017).
[2] F. Xia et al., "Graph Learning: A Survey," in IEEE Transactions on Artificial Intelligence, vol. 2, no. 2, pp. 109-127, April 2021, doi: 10.1109/TAI.2021.3076021.
[3] Wasserman, S., & Faust, K. (1994). Social network analysis: Methods and applications.

Name and workplace of master’s thesis supervisor:

Ing. Sebastián García, Ph.D. Artificial Intelligence Center FEE

Name and workplace of second master’s thesis supervisor or consultant:

Ing. Elnaz Babayeva Avast

Date of master’s thesis assignment: **24.02.2022** Deadline for master's thesis submission: **20.05.2022**

Assignment valid until: **19.02.2024**

Ing. Sebastián García, Ph.D.	Head of department’s signature	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature		Dean’s signature

III. Assignment receipt

The student acknowledges that the master’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt	Student’s signature

Acknowledgements

First and foremost, I want to express my deepest gratitude to my supervisor Sebastian Garcia, who not only supervised the thesis from start to finish but also made extra effort to help me become a better researcher, far beyond the scope of this thesis. Elnaz Babayeva, my co-supervisor, was always there to clear any miss understandings and was ready to step up when necessary. The entire Stratosphere Laboratory team was super nice and supportive, creating a wonderful and inspiring environment. I feel lucky to have been able to spend a year in this team.

It is almost impossible to show my gratitude to my family. Only because of their support, I was able to study and finish what I wanted. The thanks for everyone are not limited to this thesis and the final year of my studies, I received enormous support in everything I have aimed at. Last but not least, without my fiancee, there is no way I would be able to overcome all the pressure related to finishing my studies.

Declaration

Abstract

This thesis proposes a novel approach to computational propaganda detection. While previous research in the area of computational propaganda detection mostly focuses on either analysis of the content, or identifying malicious actors on social networks, this thesis aims to analyze if a web article is propaganda by studying its distribution pattern on the Internet. It does so by creating what we called an Article Distribution Graph (ADG) for every article in question, using various search engines, as well as social networks. The ADG represents where and how the article was referenced or duplicated. We experiment with three machine learning methods for the classification of the ADG: Graph Neural Networks, Random Forest and SVM. To evaluate the methods, we created and release a dataset *CTU-Propaganda-V1*, containing 245 articles together with their ADGs, containing data about 24,014 articles in all the ADGs combined. The 117 propaganda articles in the dataset were collected from the EUvsDisinfo database, and the 128 non-propaganda articles were manually found and verified. Results show that the best method can achieve a 81.63 % accuracy on unseen data, which is considered a very good result for the problem of detecting propaganda without text analysis.

Keywords: computational propaganda, dataset, machine learning, graph neural networks

Supervisor: Garcia Sebastian, Assist. Prof. PhD

Abstrakt

Tato práce navrhuje nový přístup k rozpoznávání výpočetní propagandy. Zatímco předchozí výzkum v této oblasti se zaměřuje buď na analýzu obsahu nebo na identifikaci škodlivých agentů na sociálních sítích, tato práce se zaměřuje na analýzu toho, zda je libovolný webový článek propagandou na základě studia jeho distribučního vzorce na internetu. Činí tak vytvořením grafu distribuce článků (Article Distribution Graph, ADG) pro každý jednotlivý článek, a to pomocí řady vyhledávačů a sociálních sítí. ADG popisuje, kde a jak byl článek odkazován či duplikován. Pro klasifikaci ADG jsme porovnali třei metodamy strojového učení: grafové neuronové sítě, klasifikátory Random Forest a SVM. Pro vyhodnocení metod jsme vytvořili a zveřejnili dataset *CTU-Propaganda-V1*, který obsahuje 245 článků spolu s jejich ADG, obsahující údaje celkem o 24 014 článcích ve všech ADG dohromady. Propagandistické články, kterých je v datasetu 117, byly získány z databáze EUvsDisinfo a 128 nepropagandistických článků bylo vyhledáno a ověřeno ručně. Výsledky prokazují, že nejlepší metoda dokáže dosáhnout přesnosti 81,63 % na testovacích datech, což považujeme za velmi dobrý výsledek rozpoznávání propagandy bez nutnosti analýzy jeho textu.

Klíčová slova: výpočetní propaganda, dataset, strojové učení, grafové neuronové sítě

Překlad názvu: Detekce propagandistických článků podle šíření na internetu

Contents

Figures

Tables

Chapter 1

Introduction

The detection of propaganda has been a priority and concern for the last 80 years [1]. Even though its importance, detecting propaganda proved to be difficult since it is a social technique with a multitude of goals, variations, and evolving techniques that adapt to different contexts and cultures. In the last two decades, the use of the Internet and social networks fostered the creation, distribution, and impact of propaganda by the help of algorithms, networks, and computers. This newer trend is referred to as *computational propaganda* and its detection has become an important priority in our societies [2].

Computational propaganda can be defined as "the use of algorithms, automation, and human curation to purposefully distribute misleading information" [3]. The automation can be used not only in propaganda distribution [4], but also in content creation [5]. Computational propaganda is hard given the amount of profiles, accounts, text, discussions, images, and videos, which makes the current manual detection of propaganda very difficult.

There have been two main approaches to detect computational propaganda with algorithms: first, text analysis to detect propaganda techniques; second, detection of the people and social profiles doing the propaganda. In the first case, natural language processing (NLP) algorithms are used to detect techniques such as *red herring* (presenting irrelevant data), *black-and-white fallacy* (presenting two alternatives as the only possibilities), and others [6, 7, 8]. In the second case, machine learning is used to automatically detect social profiles by analyzing their behavior based on some manual prior identification [9].

Even though these approaches work in their own domains (text and account profiles), news articles as web pages are still one of the most important pieces of information that the community does not yet know how to automatically identify as computational propaganda.

This thesis proposes a novel approach to detect Internet articles of computational propaganda by classifying the article distribution graph on the Internet. The article distribution graph is a graph of *who* is publishing, referencing, and *pushing* this specific article. By having one graph per article, we can apply machine learning methods on representations of these graphs to find which ones were distributed on the Internet *as propaganda* or not.

The main constraint of our work was the complete absence of a dataset of propaganda and non-propaganda articles. Therefore, we created a new and verified dataset of propaganda and non-propaganda articles together with their article distribution graphs (ADG). ADG is a graph representation of articles that are relevant to the article in question. The dataset contains 245 main articles, articles that we are interested in classifying. As part of creating the ADG for every main article in the dataset, we found and downloaded 24,014 related articles. Each propaganda article was verified using the EUvsDisinfo project [10], whereas we manually verified the non-propaganda articles.

We propose and evaluate three different machine learning algorithms to detect the distribution of propaganda in the article distribution graphs: Random Forest, Graph Neural Network (GNN), and kernel-SVM. We create feature vector for every article in an ADG, containing information about its publication date, its position in the graph using various Social Network Analysis features, article's publication date, and its level, feature created from the methodology of the ADG creation. Our results show that Random Forest was the best performing classifier with 81.63% accuracy on the test set. GNN and kernel-SVM performed worse at 75.51% and 71.43%, respectively.

The results show that classifying article distribution graphs as propaganda is possible and performs well enough. We attribute the success of Random Forest over GNN to the size of the dataset we used to train our models. Deep learning methods generally require larger amounts of data. In the future, we will explore the combination of article content analysis in the distribution graph.

The main contributions of this thesis are:

1. The creation of a new dataset of computational propaganda and non-propaganda articles, including their complete article distribution graphs.
2. A method and tool to create the ADG (Article Distribution Graph) of an article.
3. A comparison of machine learning algorithms to classify article distribution graphs.
4. A trained model that can detect if an article is propaganda or not.

Chapter 2

Theory Background

2.1 Fake News

Even though the term *fake news* has been widely used in the last decade, it is not uniformly defined. Fake news encompasses various phenomena ranging from information that is factually not true to half-true statements, misuse of images, and real news taken out of context [11]. Another definition is "fabricated information imitating news content" [12]. Note that this last definition does not describe any author's intent and can include even satiric news. The Cambridge Dictionary defines fake news as "false stories that appear to be news, spread on the Internet or using other media, usually created to influence political views or as a joke" [13].

Fake news has been the most important focus of research and detection in the last decade for two main reasons. First, it is mostly text. Second, there are studies defining the philosophical traits of fake news in order to identify them, such as appealing to emotions rather than facts. This means that there is a difference in the text style that suits a myriad of text-based machine learning techniques for propaganda detection.

Fake news can be further classified based on its intent to misinformation and disinformation. *Misinformation* describes false information that is not intended to mislead the reader. It is often spread because the author is ill-informed or does not have a perfect memory. *Disinformation*, however, is false information spread deliberately to mislead and influence the reader [14].

2.2 Propaganda

Propaganda is a very different concept compared to fake news. The Cambridge Dictionary defines propaganda as "information, ideas, opinions, or images, often only giving one part of an argument, that are broadcast, published, or in some other way spread with the intention of influencing people's opinions" [15]. This definition is important for our research, since we are aiming at detecting the spread on the Internet.

Propaganda may use fake news [14], or it may not. There are many techniques used in propaganda. The Institute for Propaganda Analysis (IPA) in 1937 described seven techniques [16], including *name calling*, attaching negative label to a person so the reader rejects his idea without examining the content; *card stacking* - presenting only one opinion, omitting the other opinions and unfavourable statistics); and *bandwagon*- encouraging action by the reader by persuading him everyone else is doing it [17]. However, propaganda is much more than these techniques. Propaganda needs a coordinated effort, a multitude of media, and a constant exposure of the audience to the message. These characteristics is what makes propaganda a good candidate to detect such coordination and large effort in this thesis.

2.2.1 Computational Propaganda

Computational propaganda is a term describing the usage of automation for propaganda purposes. The automation can be either by machines or by a large group of people. The most common use of the automation is in the spread of propaganda messages by coordinated human effort, often referred to as "troll armies" [18], the use of automated bots to post on social media [19] to flood of opinions in social networks, the automatic generation of images, videos and text. Moreover, since machine learning models like GPT-3 [20] and PaLM [21] can produce human-like text content, it is likely that algorithms may produce and distribute propaganda content in the near future. We

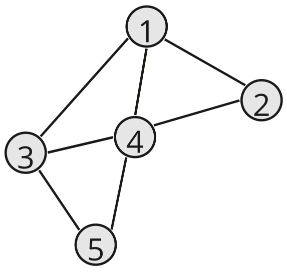

Figure 2.1: Simple graph. Set of nodes $V = \{1, 2, 3, 4, 5\}$ and set of all edges $E = \{\{1, 2\}, \{1, 3\}\{1, 4\}\{2, 4\}, \{3, 4\}, \{3, 5\}, \{4, 5\}\}$.

believe that any content produced by algorithms might be detectable by algorithms.

2.3 Graph

An oriented graph G is a tuple $G = (V, E, \epsilon)$, where V is a finite set of nodes, E is a finite set of edges. ϵ is a mapping assigning oriented pair of nodes for every edge.

$$\epsilon : E \to \{(u, v) \mid u, v \in V\}. \tag{2.1}$$

Nodes $u, v \in V$ are called *adjacent* if there is an edge $e \in E$ such that $\epsilon(e) = (u, v)$. There are many options how to represent a graph. *Adjacency matrix* A of a graph G is a matrix of shape $|V| \times |V|$, where $A_{ij} = 1$ if there is an edge from node i to node j, else $A_{ij} = 0$. Adjacency matrix of a graph in Figure 2.1 is

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \tag{2.2}$$

An unoriented graph can be created from an oriented graph The adjacency matrix A is always symmetric for an unoriented graph, whereas for oriented graphs it might not.

A *path* from u to v is a sequence of unique nodes and edges $p = (v_1, e_1, v_2, \ldots v_{n+1})$

where $v_1 = u$ and $v_{n+1} = v$ and $\epsilon(e_i) = \{v_i, v_{n+1}\}$. *Length* of a path $c(p)$ is the number of edges used in the path. *Shortest path* is a path $v_1, e_1, \ldots, v_{n+1}$ with the smallest length. The length of the shortest path is called distance, denoted as $d(u, v)$.

A degree $deg(u)$ of a node $u \in V$ is the number of edges oriented from and to the node.

2.4 Random Forest

Random forest [22] is a Machine Learning (ML) ensemble method that combines many decision trees. The method can work with any type of data represented as a feature vector, regardless of, the range of individual features, which makes it a powerful tool for working on data such as categorical variables. Decision tree partitions data into discrete subsets using sequential set of rules. Unseen data can be classified using the subset, most often as the majority class represented in the subset.

If we represent the input vector in a traditional form, where x is a feature vector, and Y is the label:

$$(\mathbf{x}, Y) = (x_1, x_2, \ldots, x_i, Y) \tag{2.3}$$

And the whole set of input vectors is called dataset D. The learned predictor goes from the dimensional space of x to Y:

$$h : \mathbf{x} \rightarrow Y \tag{2.4}$$

Decision tree algorithms are known to suffer from overfitting. Therefore, the Random Forest classifier was proposed to reduce the overfitting tendency of a decision tree. It does so by sampling with replacement multiple sets and learning a decision tree for every set. In addition, only a randomly selected subset of all features is considered when searching for the splitting feature. Classification is performed by using every tree to predict, and then use a majority vote using these predictions.

The training of each tree is performed in recursive splits of the dataset D to grow a tree. Starting in the root node, the algorithm randomly picks a feature x_i and splits the dataset into two parts D_1, D_2. The sets D_1, D_2 are added as children of the root node, and are considered roots of their own subtrees. The split is performed recursively until either every leaf partition contains at most *leaf size* samples or it reaches the *maximum depth*. Both these hyper-parameters have to be chosen.

In a node with dataset D, the split is performed by thresholding along a feature x_i. The feature x_i as well as the threshold θ is chosen so it minimizes the Gini Impurity of the sets D_1, D_2. Suppose K classes, p_i is fraction of samples belonging to class i. Gini Impurity is defined as

$$I(D) = 1 - \sum_{i=1}^{K} p_i \tag{2.5}$$

The value is minimized when all samples in set D belong to single class j, therefore $p_j = 1$ and $I(D) = 0$.

When using the learned tree for classification, it starts at the root node and follows the split rules, until it reaches a leaf. The sample is classified using a majority rule using that leaf set.

2.5 Graph Neural Networks

Given that the data for our research consist of distribution graphs of articles on the Internet, we explored the use of graph based neural networks for its classification.

Graph Neural Networks (GNN) are neural networks operating on graphs. It does so by keeping and updating node and edge embeddings. In every layer, an embedding is updated with the information from neighbouring nodes and edges. There are three main types of tasks that GNNs can solve:

1. Node classification. The task is to determine the labeling of nodes by looking at properties of nodes in their surrounding. The task might be prediction of a paper given its words and citation network.

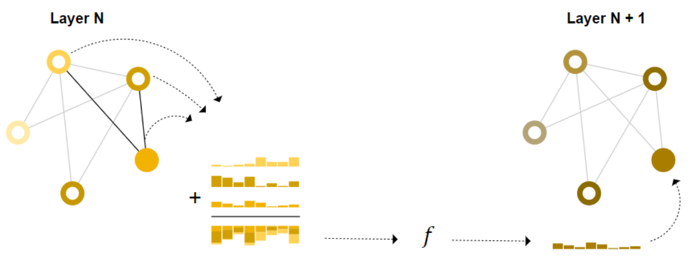

Figure 2.2: Message passing of GNN layer one layer. Node embeddings from neighbouring nodes are aggregated (summed). The output embedding is transformed using an activation function f on the aggregated vector. Figure from [23]

2. Graph classification. This task is to classify entire graph. One very common problem is predicting whether chemical molecule is toxic or not.
3. Link prediction. This algorithm has to understand the relationships between nodes in a graph and find unknown relationships or predict future relationships. It is essential in social networks and suggesting possible friends to the users.

Like in other neural networks, GNNs have many architectures available. The core of every GNN layer is how message passing is performed. Given graph $G = (V, E)$ and node representation $\boldsymbol{h}^0$, the node representation h^{l+1} after layer l is calculated as

$$h_i^{l+1} = f(g(W^l h_i^l \cup (\bigcup_{j \in N_i} W^l h_j^l))) \tag{2.6}$$

where, W is a trainable weights matrix, N_i denotes neighbourhood of node i, g is a permutation invariant aggregating function, usually *sum*, f is an activation function and W^l is a weight matrix to be learned.

Message passing can be expressed in matrix form using the adjacency matrix. For example, when aggregation function is *sum*, the message passing can be formulated as

$$H^{l+1} = f(\hat{A} W^l H^l) \tag{2.7}$$
$$\hat{A} = A + I, \tag{2.8}$$

where A is adjacency matrix, I is the identity matrix and f is activation function. An example visualization of message passing in one layer is visualized in Figure 2.2.

For the task of graph classification, graph is processed for For the task of graph classification, the graph $G = (V, E, H)$, where H is node initial node embedding, is processed using graph convolution layers. After k layers, a single vector $\psi(G)$ is created as mean pooling of the node embeddings:

$$\psi(G) = \frac{1}{|V|} \sum_{j \in V} h_j^k \tag{2.9}$$

This vector is than processed using Linear layers to make a prediction.

Chapter 3

Related Work

This chapter first reviews the existing datasets for fake news and propaganda detection. Then it shows two approaches to detecting propaganda and fake news: text analysis perspective and network analysis perspective.

3.1 Available Datasets

One of the first benchmark datasets in the propaganda detection area, called *trusted, satire, hoax, propaganda 2017* (TSHP-17) was published by Rahskin et al. [6]. It consists of a corpus of 22.5k articles collected from 11 news outlets. Every article was assigned one of four classes: trusted, satire, hoax, or propaganda. The dataset was created using distant supervision: the article is assigned a class based on the label given to the publisher's outlet by the US News & World Report[1]. Barrón-Cedeño et al. [7] released a new dataset QProp in reaction to what they believed to be a drawback of TSHP-17: given the small number of outlets per class used, the systems trained on TSHP-17 might be modeling news outlets rather than propaganda. QProp consists of 53.3k articles from 104 news sources. Once again, they used distant supervision with a news source labeled by Media Bias/Fact Check (MBFC)[2]. They labeled every article as propaganda or non-propaganda, with about 10%

[1]https://www.usnews.com/
[2]https://mediabiasfactcheck.com/

articles labeled as propaganda.

LIAR dataset published by William Yang Wang [24] stores over 12,000 short statements labeled by Politifact[3] to one of 6 classes based on how truthful the statement is.

Since propaganda uses specific rhetoric and techniques, Da San Martino et al. [25] created a dataset PTC to detect manipulative spans of text. The corpus consists of 451 articles, each manually accessed by professional annotators. The annotation is performed at fragment level: specific text spans rather than full documents. Each flagged text fragment is assigned one of 18 propaganda techniques, such as *loaded language* (using phrases with strong emotional implications), *doubt* (questioning the credibility of something), *red herring* introducing irrelevant information to the issue being discussed to drive away attention). In the similar manner PTC-SemEval20 [26] was created for the Lexical and Computational Semantics and Semantic Evaluation (SemEval) [27].

Baisa et al. [8] published a Czech News dataset. They collected 7,494 Czech news articles and selected 18 propaganda techniques, among which is *blaming* (when the text accuse someone of something), *emotions* (what emotion is the text trying to evoke), *expert* (is the text supported by an expert). Every article is labeled for existence of every propaganda technique.

Mitra & Gilbert [28], tracked over a billion tweets over the course of three months in 2015. From that, they curated a set of over 60 million tweets grouped into 1,049 real-world events, creating the CREDBANK dataset. Every tweet is assigned credibility score by over 1,700 people.

FacebookHoax dataset by Tacchini et al. [29] includes networking information. A total of 15,500 facebook posts, from July to December 2016, were labeled as either hoax or non-hoax. Information about over 90,000 user engagements in the form of likes is included.

Most of the publicly available datasets contain purely the textual content and its label. Labels are either document-wise or fragment-wise. The datasets that contain some networking information are restricted to social networks. In this thesis, we want to classify any internet article, therefore we require dataset

[3]https://www.politifact.com/

with propaganda and non-propaganda URLs. Such dataset of propaganda articles is absent.

3.2 Text Analysis Perspective

Psycho-linguistic work has shown that speech patterns can be signs of a speaker trying to obscure the truth [30]. The existence of different patterns in manipulative and truthful texts allows for detecting propaganda and non-propaganda by purely analyzing the text.

The first works attempted to detect deceitful statements. Ott et al. [31] analyzed reviews on TripAdvisor[4]. They used SVM and Naive Bayes classifiers on a set of features extracted using *Linguistic Inquiry and Word Count* software [32] to detect if reviews were automatically generated. Mihalcea & Strapparava [33] used tokenization and stemming, NLP techniques, to extract features used for the same classifiers, SVM and Naive Bayes.

Rashkin et al. [6] classified entire texts. Da San Martino et al. created the system Proppy [7], which works using n-grams as well as additional features such as capturing style, vocabulary richness, and readability. Baisa et al. [8] transformed text to vector using used TF-IDF weighted Bag-of-Words and classified using 10 different classifiers, including SVM and Random Forest.

Da San Martino et al. [25] proposed two tasks for propaganda detection: *Sentence-level Classification (SLC)*, given a sentence, decide whether it uses at least one propaganda technique, and *Fragment-level classification(FLC)*, identify both span and the propaganda technique in the text. They also propose various BERT [34] based models for their tasks.

For the task of FLC, Yosuf et al. [35] and [36] used fine-tuned BERT. Gupta et al. [37] use a LSTM-CRF model [38] for the SLC task, by proposing an ensemble method combining logistic regression, Convolution Neural Network and BERT.

Even tough these techniques had good results, this thesis does not use

[4]https://www.tripadvisor.com/

explicit text content to classify the web articles.

3.3 Network Analysis Perspective

Early works aimed at detecting and classifying individual nodes in a social network as legitimate or adversary. Clusters of adversary nodes were considered to work in coordination to achieve their target goal [39]. It was shown that bots can be used to influence public opinion in favour of a politician [40]. Companies use social media for communicating with the users, advertising and marketing [41]. Automated accounts, either hijacked accounts of real users, or adversary-owned, can mimic actions of real users to influence public opinion and recommending algorithms [42].

Cao et al. [43] introduced SybilRank, a tool that analyzes social graph properties to estimate the likelihood of being fake. Yang et al. [44] targeted the same problem, however, they focused on describing features of the profiles, such as frequency of clicks performed by these accounts.

Deep neural network approaches were used by Kudugunta & Ferrara [45], who created an LTSM based architecture capable of detecting a bot from a single tweet. Wu et al. [46] proposed an improved conditional generative adversarial network to extend imbalance sets, which allowed them to better train neural network classifiers.

Six different teams participated in the DARPA Twitter Bot Challenge [47]. It was shown that supervised machine learning approaches alone were insufficient due to the lack of training data. All participating teams analyzed syntax and semantics of users' tweets and its evolution over time. They included user profile features, such as user's photo, and network features, such as the deviation of the sentiment score of the user from his followers and accounts the user followed.

All previous approaches focused on detecting a single fake user in a social network without considering their coordination. An alternative approach is to assume that bots are working together aiming to achieve a shared goal. This coordination can be used to detect a group of users in an unsupervised way to overcome the shortcomings of supervised algorithms due to the limited availability of training datasets [48].

Systems for detecting entire groups of users aim to detect connectivity patterns. Coordinated efforts are seen as almost fully connected sub-graphs. Hooi et al. presented BIRD [49] for detecting automated reviews based on clustering of reviewers. Chetan et al. [50] created CoReRank, an unsupervised system focused on retweeting patterns and tweet content.

Chapter 4

Dataset

The goal of this thesis is to analyze the distribution patterns of a web article. To find the pattern, we require URLs to obtain the features that we need. As described in Section 3.1, there is no such dataset for propaganda detection.

This chapter describes how our dataset was crated. We start by discussing the methodology for collecting articles. Next, we discuss the gathering of the dataset. The dataset contains 245 web articles, 128 of them are non-propaganda articles, and 117 are from propaganda articles.

This thesis introduces the concept of Article Distribution Graph (ADG). An ADG is a way to represent the distribution of an article on the Internet. By *article*, we understand a news article on the Internet with a unique URL address. The ADG contains the *main* article, and the Internet articles that *reference back* to the *main* article. So, for example, if the article *https://example.com/news.php?id=2345* is linked from *https://different.com/things.php?id=5553* then the second article will be included in the ADG of the first. ADG are more complex than this, and a comprehensive explanation can be found in Subsection 4.2.

For each article, part of the features extracted are: URL, title, second level domain domain, publication date, HTML content and human-readable textual content.

4.1 Collecting Propaganda and Non-propaganda Articles

For our task, we require URLs of both propaganda and non-propaganda articles. We aim to find articles that we can be certain are or are not propaganda.

There are many non-profit organizations that detect disinformation and propaganda, *e.g.*, Politifact[1], Čeští elfové[2]. Usually, they select a statement and fact check every claim and provide explanation with background information. We used data provided by the EUvsDisinfo project [10].

EUvsDisinfo is a project of the European External Action Service[3]. It was established to address the Russian Federation's propaganda affecting states of the European Union. Their core objective is to increase public awareness of Kremlin propaganda so that European citizens can develop resistance to manipulation. They define their work as "Using data analysis and media monitoring services in 15 languages, EUvsDisinfo identifies, compiles, and exposes disinformation cases originating in pro-Kremlin media that are spread across the EU and Eastern Partnership countries." [10].

There is a publicly available database of more than 1,200 samples of all the propaganda articles they have examined on their website. Each entry consists of a summary of the article, a detailed disproof, including the sources of their claim, and the URL. We collected 500 articles from their database dated from June to August 2021.

Articles that are non-propaganda are more challenging to find. It is not clear what exactly is not propaganda. Any published article might be propaganda; however, people may not have detected it. To the best of our knowledge, no organization verifies that an article is not propaganda. Some organizations, such as Politifact, review various statements and evaluate their truthfulness. However, showing that the article is true does not mean that it is not propaganda.

[1]https://www.politifact.com/
[2]https://cesti-elfove.cz/
[3]https://www.eeas.europa.eu/

To avoid introducing additional biases into the data, we keep both the topic and geographical location of chosen articles similar to the collected propaganda articles. It is likely that the distribution patterns look different with respect to geographical location. Any model trained on such a dataset might learn to differentiate between the location, rather than propaganda and non-propaganda, similarly to what Rashkin et al. [6] found, their model differentiated between news sources rather than the content.

To maintain the geographical similarity between the propaganda or non-propaganda articles, we searched for Russian or Eastern European news articles. This introduced a new complication in the form of a language barrier. We had to main rely on Google Translate[4] to read the content of the articles, however the supervisor-specialist Elnaz Babayeva speaks Russian natively and could verify the data. To maintain the topic similarity of the EUvsDisinfo articles, we had to find politically themed articles.

The verification that an article is non-propaganda was done in two steps:

1. We checked the writing style of the article and whether it uses any propaganda techniques. Since we used Google Translator for translation, it may happen that the propaganda technique was lost in translation.
2. We fact-checked the main content of the article. If the article discussed multiple events, we discarded such an article due to the difficulty of fact-checking.

Using the above methodology, we collected 128 non-propaganda articles.

4.2 Building Article Distribution Graphs

The creation of the dataset starts with what we called the *main* article. This is the article that is verified to be propaganda or not-propaganda and that we want to see how it was distributed. For each main article we create one ADG (Article Distribution Graph).

[4]https://translate.google.com/

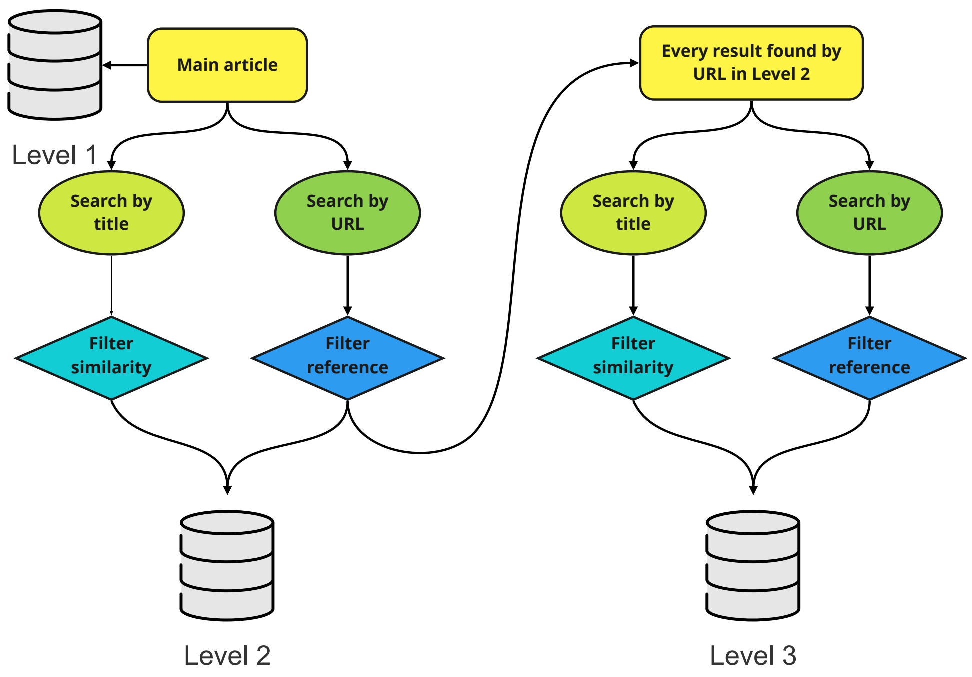

Figure 4.1: Methodology of the Article Distribution Graph creation. To create ADG of the main article, its title and URL is used as a keyword for searching using engines, as well as social networks. The irrelevant results are filtered out, the relevant are stored. All relevant results that were found by URL are used to perform the search again.

We create the Article Distribution Graphs by searching the main article on the Internet. The diagram of how the ADG is created is shown in Figure 4.1.

The ADG represents the graph of all the articles that reference or share content with the main article. An example of an ADG is shown in Figure 4.2. Notice that the links by URL and Title are the opposite of HTML links. In an HTML link (commonly referred to as *link*) an article *a links* to other article *b* using HTML tags, such as the <*a*> tag. However, in our ADG URL and Title relationships, we want to know that article *b* referenced article *a*. This is much harder since there is no list of web pages that *are linked from* other web page. There is no *back-link* in HTML.

4.2.1 Internet Searching

We search the Internet to find from which articles the main article is referenced or its content is duplicated. Every search is performed simultaneously on four

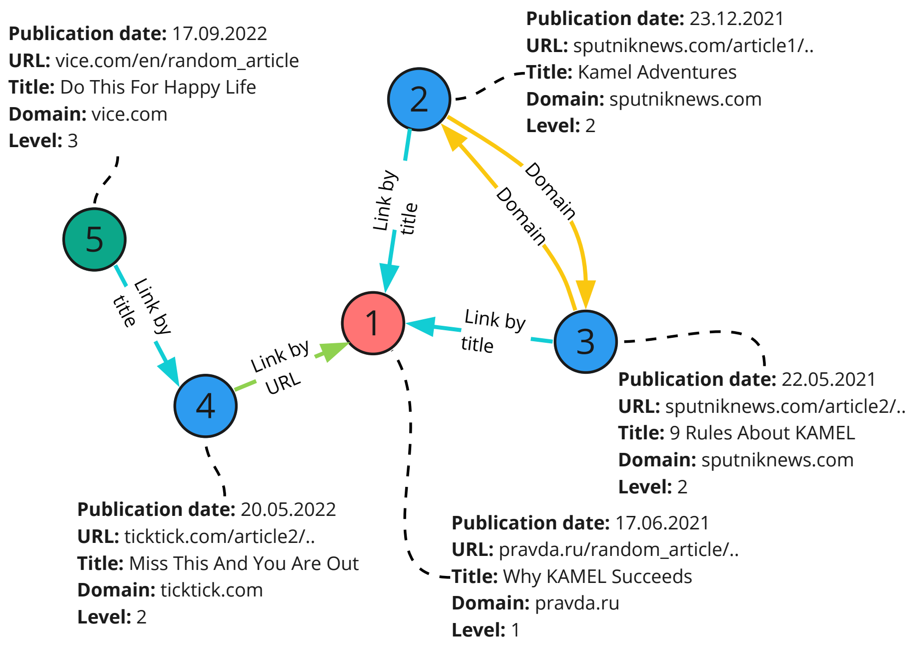
Publication date: 17.09.2022
URL: vice.com/en/random_article
Title: Do This For Happy Life
Domain: vice.com
Level: 3
Publication date: 23.12.2021
URL: sputniknews.com/article1/..
Title: Kamel Adventures
Domain: sputniknews.com
Level: 2
2
Link by title
Domain
Domain
5
Link by title
1
Link by title
3
Link by URL
4
Publication date: 22.05.2021
URL: sputniknews.com/article2/..
Title: 9 Rules About KAMEL
Domain: sputniknews.com
Level: 2
Publication date: 20.05.2022
URL: ticktick.com/article2/..
Title: Miss This And You Are Out
Domain: ticktick.com
Level: 2
Publication date: 17.06.2021
URL: pravda.ru/random_article/..
Title: Why KAMEL Succeeds
Domain: pravda.ru
Level: 1

Figure 4.2: Example of Article Distribution Graph. Every node corresponds to an Article, it store publishing date, URL, Title, Domain and Level. Node 1 is the main article, it is linked by title by articles 2 and 3, which have the same domain. Article 4 links the main article by URL, therefore it is used for the next level search, and article 2 and 3 are not used for the next Level search because they are found by title. Node 5 links 4 by title.

of the most popular search engines: Google, Bing, Yandex and Yahoo, as well as two of the most popular social networks in Eastern Europe: Twitter and VKontakte.

For every article, we search twice with different keywords. Firstly, we *search by title* of the article, secondly, we *search by URL*. All articles returned from the search engines (as well as social networks) when searching by title keyword are referred to as *found by title*. Similarly, articles returned from search engines when searching by URL are referred to as *found by URL*.

4.2.2 Filter Search Engines Result

Before adding the results to the ADG, we need to filter unwanted results. All search engines find many unwanted articles, therefore we need to keep only the relevant results. We download the HTML source for every result and from it extract the clean article content. We filter the results found by title differently to the ones found by URL.

If a result was found by title of an article, we compare the article's content to the content of the result. We evaluate the similarity of the content using the Levenshtein Ratio, a similarity metric of strings a and b with lengths $|a|, |b|$, respectively, defined as:

$$lev\_ratio(a,b) = 1 - \frac{lev(a,b)}{max(|a|,|b|)} \tag{4.1}$$

where $lev(a,b)$ is Levenshtein distance:

$$lev(a,b) = \begin{cases} |a| & \text{if } |b| = 0 \\ |b| & \text{if } |a| = 0 \\ lev(tail(a), tail(b)) & \text{if } a[0] = b[0] \\ 1 + min \begin{cases} lev(tail(a), b) \\ lev(a, tail(b)) \\ lev(tail(a), tail(b)) \end{cases} & \text{otherwise} \end{cases} \tag{4.2}$$

where $tail(x)$ is all but the first character of string x and $x[0]$ is its first character. The final distance is the minimal number of *operations* required to perform on the strings to transform one string into the other. *Operation* can be one of the following: delete a character, insert a character, or modify a character.

For every result found we calculate the Levenshtein Ratio of the article's clear content $cont\_r$ and clear content of the article $cont\_a$ used for searching, and we keep only the results if $lev\_ratio(cont\_a, cont\_r) \geq \omega$.

We learned ω by searching by title for 40 different origin articles. As a result, we extracted 1,153 articles and found the best threshold to be $\omega = 0.3853$ with 71% accuracy. This means that the content is similar enough that it can be considered to share substantial part of the content.

If a result was found by URL, we need to check whether the HTML code of the result contains the URL of the article. If there is the URL, it means that there is a direct reference from the result to the article.

4.2.3 Multi-level graph creation

At the beginning of the creation of the ADG, we have only the main article to which we want to create the ADG. We store the article in a set called Level 1. Every result found by title or by URL and passing the respective filtering is stored in Level 2. We repeat the search for all articles found by URL and stored in level 2. Again, after the results passes the filtering, we store them in Level 3.

We repeat the search for articles in Level 2 only found by URL. The reason is that when searching for title, we compare the similarity of the textual content. If we search again using Level 2 article found by title, the textual similarity might be vastly different. Articles found by URL, however, contain direct reference to the article. We assume this direct reference is stronger, therefore, searching again using this article finds relevant articles.

Chapter 5

Implementation of a Tool to Create our Dataset

For this thesis, we extended the Web Reverse Graph Extractor (Werge), a tool developed by Garcia & Babayeva [51]. Werge creates the ADG described in Section 4.2 from the URL of an article. In this chapter, we describe Werge and its components.

Werge generates a SQLite database as an output. The database contains two tables: a table with information about the articles and a table with links between the articles.

The table with information about the articles stores the following information:

- Article's URL - where the article is hosted
- Article's domain - domain of the article's URL
- Article's publication date - extracted the date when the article was published
- Article's level - the shortest path from the article to the origin article
- Article's HTML source - a raw HTML source code

- Article's content - human readable text of the article, extracted from HTML page

The table with information about the links between the articles stores the following information:

- Article used for searching
- Article found
- Type of search, either URL or Title

Werge has two main functionalities: creating the Article Distribution Graph structure, and performing the data extraction about the article.

5.1 Creating Article Distribution Graph Structure

To create the ADG, when searching by title or by link, we use four different search engines: Google, Bing, Yahoo, Yandex. And we search for public content on Twitter and VKontakte.

Searching in search engines is performed by Google-Search-API created by SerpApi [52]. SerpApi is a paid service; however, they allowed us to use their API for free for research purposes. Searching for *Monkey* using Google and getting results in the form of a dictionary using SerpApi Google-search-API is as simple as:

```
from serpapi import GoogleSearch

search_parameters = {
    "engine" : "google",
    "q" : "Monkey",
    "api_key" : SECRET_API_KEY
    }
results = GoogleSearch(search_parameters).get_dict()
```

There are many optional parameters available, such as search engine, page number, number of results to find, *etc.*. The parameters vary slightly for different search engines.

Algorithm 1: Werge main algorithm

```
   Result: Store all data and links into a database
 1 create_database()
 2 level[1] = [query_url]
 3 for current_level in [1,2,3] do
 4     urls_to_search = []
 5     for url in level[current_level] do
 6         article = get_content(url);
 7         urls_found_by_url = search(article["URL"])
 8         for found_url in urls_found_by_url do
 9             found_article = get_content(found_url)
10             if contains(found_article["HTML"], article["URL"]) then
11                 urls_to_search.append(found_article["URL"])
12                 save_data(found_article)
13                 save_link(found_article, article, "url", current_level)
14             end
15             if article["title"] is not None then
16                 urls_found_by_title = search(article["title"])
17                 for found_url in urls_found_by_title do
18                     found_article = get_content(found_url)
19                     if filter_similarity(found_article["content"], article["content"]) then
20                         save_data(found_article)
21                         save_link(found_article, article, "title", current_level)
22                     end
23                 end
24             end
25         end
26         level[current_level+1] = urls_to_search
27     end
28 end
```

level

The next block starting from Line 15 is very similar to processing search by URL, with two differences. Again, we perform a search, however, now we use *title* as search keyword. Next, we compare the similarity of contents of searched URL and found URL with the function *filter_similarity(found_url_data, url_data)*. The similarity is measured as Levenshtein Ratio of clear contents, contents extracted using BeautifulSoup [53].

The function *filter_similarity* uses the library Levensthein[1] and filters out all results if the Levenshtein Ratio is smaller than $\omega = 0.3854$

```python
import Levenshtein

def filter_similarity(data1, data2, omega):
    content1 = data1["content"]
    content2 = data2["content"]
    return Levenshtein.ratio(content1, content2) >= omega
```

5.2 Data Extraction

We aim to extract the following information for every article processed:

- URL of the website
- HTML content of the website
- Title of the article
- Article text
- Publication date

At the beginning of processing any article, only its URL is available. Using the Python library Requests, all the HTML source code is downloaded. The HTML source code is parsed by a library BeautifulSoup4 (BS4). BS4 is a Python library for pulling and parsing data from HTML files. Using this library, we extract both the title and the text.

There are many libraries that can extract the publication date of a website, however none of them works particularly well. We experimented with several

[1]https://pypi.org/project/python-Levenshtein/

Date extraction method	TP	FP	TN	FN
First Date Present	46	35	2	1
Heuristic	56	41	3	1
Newspaper3k	56	1	2	31
Htmldate	82	43	7	13
Newspaper3k→Heuristic	92	34	1	0
Google-Search-Api→Newspaper3k→Htmldate	124	17	7	1

Table 5.1: True Positive (TP) represents the approach finds a date and it is the correct one, False Positive (FP) is when the approach extracts a date, however it is wrong. True Negative (TN) is the case when the approach is unable to extract date and there is no date in the HTML source code. False Negative (FN) for being unable to extract a date, however, it is present in the source.

of them as well as developing our own method to find the best approach for date extraction. First Date Present is simple method to find the first string in a form of a date in the HTML source code. Heuristic approach is selecting the date closest to the title in terms of char-distance to the title„ because in the most of the cases the publication date is placed next to the title. We find all strings in a form of a date and select the one which is placed closest to the title. Newspaper3k is a library for browsing news sites as well as extracting data, including the publication date. Htmldate [54] is a library for extracting dates of website. Google-Search-API is a service for using search engines, where the results also have publication date. We also included combined approaches, use one library to extract the date and if it does not extract anything, use different approach. For this we combined Newspaper3k with Heuristic approach and Google-search-API with Newspaper3k and Htmldate. We extracted the publication dates of many websites and each approach on many websites and manually checked whether the date is correctly extracted Results of our manual analysis is in Table 5.1.

In our manual review, the best performing method was to use the combination of Google-Search-API, Newspaper3k[2] and HTMLDate[3]. First we search the date by Google-Search-API, if it returns no date, use Newspaper3k and if it does not extract the date, use Htmldate yielding 82%. The approach can be summarized as follows:

```
publication_date = extract date using Google-Search-API
if publication_date is None:
    publication_date = extract date using Newspaper3k
```

[2]https://newspaper.readthedocs.io/
[3]https://pypi.org/project/htmldate/

```
if publication_date is None:
    publication_date = extract date using HTMLDate
```

Chapter 6

Methods to Detect Computational Propaganda

We use machine learning (ML) methods to create a model that predicts whether a URL is propaganda or not. The task is defined as a binary graph classification. Our goal is to create a predictor m:

$$m : ADG \rightarrow \{propaganda, non\text{-}propaganda\} \tag{6.1}$$

We measure the performance of a model m using the accuracy metric:

$$Accuracy = \frac{1}{|D|} \sum_{(g,y)\in D} [\![m(g) = y]\!] \tag{6.2}$$

where D is a dataset, (g, y) is the ADG and its label, either propaganda or non-propaganda and $m(g)$ is the output of the model.

6.1 Features Available

The ADG is an orientated graph with features for every article and edge. We expand descriptions of nodes in the ADG by adding additional features. A node in the graph represents article as well as other node features as:

1. Level of the article
2. Time difference
3. Missing time difference
4. In-degree centrality
5. Out-degree centrality
6. Closeness centrality
7. Eigenvector centrality
8. Clustering coefficient

All four centralities as well as the clustering coefficient are measurements from Social Network Analysis and are described in Section 6.1.1. These SNA features are denoted as V_{SNA}. We also denote V_{TD} the time difference as well as the missing time difference and V_L the level feature.

Time difference is a feature created from the publication dates. This transforms publication date into a numbers and makes the feature relative to node v, which is the node used for searching and found u.

$$TD(u) = \begin{cases} (date(v) - date(u)).days & \text{if publication dates of u,v are known,} \\ 0 & \text{if at least one publication date is unknown} \end{cases} \tag{6.3}$$

In addition to the Time Difference feature, we introduce the Missing time difference MTD to track if the second case of Equation 6.3 occurred.

$$MTD(u) = \begin{cases} 0 & \text{if u is original article's node} \\ 0 & \text{if publication dates of u,v are known,} \\ 1 & \text{if at least one publication date is unknown} \end{cases} \tag{6.4}$$

We also create seven binary features edges. Every feature connect two nodes based on whether the condition is true:

- E_u Found by URL
- E_t Found by title
- E_d Share the same domain
- E_{ru} Reversed edge of found by url
- E_{rt} Reversed edge of found by title
- E_a Edge exists if the two pairs of node were found by URL, found by title, share domain, $E_a = E_u \cup E_t \cup E_d$
- E_{ra} Edge exists if the two pairs of node were found by URL, found by title, share domain, reversed found by URL, reversed found by title, E_{ra} $= E_u \cup E_t \cup E_d \cup E_{ru} \cup E_{rt}$

6.1.1 Node Features: Social Network Analysis

Social network analysis is a process of analyzing graphs using graph theory methods. We selected five metrics describing positions of a node in the graph.

In Degree Centrality

In degree centrality of node $u \in V$, denoted $deg^{-}(u)$, is mathematically defined as

$$deg^{-}(u) = \frac{1}{|V|-1} \sum_{x \in V} [\![(x,u) \in E]\!] \qquad (6.5)$$

where V is a set of nodes in the graph and $[\![(x,u) \in E]\!] = 1$ if $(x,u) \in E$, $[\![(x,u) \in E]\!] = 0$ if $(x,u) \notin E$. The in-degree centrality of a node u measures how many edges are oriented to u. The value is normalized by $|V|-1$. Note that in a graph without parallel edges and without loops, there are at most $|V|-1$ possible edges connected to the node.

Out Degree Centrality

Out degree centrality is very similar to in degree centrality. The difference is that we calculate number of edges oriented from a node u.

$$deg^{+}(u) = \frac{1}{|V|-1} \sum_{x \in V} [\![(u,x) \in E]\!] \tag{6.6}$$

Closeness Centrality

$$clo(u) = \sum_{x \in V} \frac{|V|}{d(x,u)} \tag{6.7}$$

Closeness centranlity shows how the node is in the center to the graph, if it is on average close to all the other nodes or not. Note that $d(x,u) = \infty$ if there is no path from x to u. For calculating closeness centrality it is assumed that $\frac{|V|}{\infty} = 0$.

Eigenvector Centrality

Eigenvector centrality calculates the *importance* of the node based on how many important nodes it links to. This measurement

$$A\mathbf{x} = \lambda\mathbf{x} \tag{6.8}$$

where A is adjacency matrix λ is its largest eigenvalue and $\mathbf{x}$ is corresponding eigenvector. Following Perron–Frobenius theorem the largest eigenvalue of a matrix is unique and positive if the matrix has only non-negative entries. Adjacency matrix satisfies this requirement. The eigenvector centrality of node i is the i-th element of vector $\mathbf{x}$.

We are calculating the centrality on the reversed graph because of the real life meaning. As stated above, eigenvector centrality calculates the *importance* of the node based on how many important nodes it links to. On the reversed ADG, the higher the eigenvector centrality of a node, the more it is referenced by often referrenced nodes.

Clustering Coefficient

This measurement describes information about neighbours of node u. The value is calculated on unoriented graphs. The idea is about measuring how many of u's neighbours are connected, therefore measuring how many triangles the node is a part of with respect to the number of possible triangles including node u

$$clu(u) = \begin{cases} 0 & if deg(u) \leq 1 \\ \frac{2T(u)}{deg(u)(deg(u)-1)} & if deg(u) > 1 \end{cases} \tag{6.9}$$

where $T(u)$ is number of triangles u is a part of.

6.2 Models

We tested three different algorithms. First, we hand-crafted a feature vector to describe each graph and use random forest for the classification. Second, we tested kernel-SVM with graph kernel methods, a similarity measurement methods based on the graph structure. Last, we used a Graph Neural Network to combine both hand-crafted features with the graph structure.

6.2.1 Random Forest

To use random forest, a feature vector representing a graph is required. We create following features and concatenate them into a single feature vector.

1. Fraction of nodes in level l for every level. $\frac{1}{|V|}\sum_{i\in V}[\![level(i) = l]\!]$
2. Mean SNA features, create the SNA features for every node and calculate their node-wise mean. $\frac{1}{|V|}\sum_{i\in V} v(i)$ where v denote the SNA feature.
3. Mean time difference of nodes $\mu_{TD} = \frac{1}{|V|}\sum_{i\in V} TD(i)$
4. Standard deviation of the time difference $\frac{1}{|V|}\sum_{i\in V}(TD(i) - \mu_{TD})^2$
5. Mean missing time difference $\frac{1}{|V|}\sum_{i\in V} MTD(i)$

6. Number of nodes in the graph $|V|$
7. Number of edges in graph $|E_d| + |E_u| + |E_t|$
8. Fraction of links found by URL over all links found $\frac{|E_u|}{|E_u|+|E_t|}$
9. Fraction of nodes having unique domain $\frac{1}{|V|}\sum_{i\in V}[\![\sum_{j\in V} domain(j) = 1]\!]$

These features are concatenated into a feature vector of dimension 15.

6.2.2 Graph Neural Network Architecture

For Graph Neural Network application we use Relation Graph Convolution layer (RelGraphConv) proposed in [55]. This layer allows multiple types of edges and learns different transformation for every edge type. Message passing in RelGraphConv layer is performed as

$$h_i^{t+1} = f(W_0^l h_i^l + \sum_{r\in R}\sum_{j\in N_i^r} \frac{1}{c_{i,j}} W_r^l h_j^l) \tag{6.10}$$

where R is a set of all edge types and N_i^r is a neighbourhood of node i considering edge type r, nodes j such that $(j,i) \in E_r$. $c_{i,r}$ is a normalisation term set as $c_{i,j} = \sqrt{deg(i)deg(j)}$. The advantage of this GNN is that we could utilize our different edge types in the graph.

We created a architecture consisting of two Relation Graph Convolution layers and one Linear layer seen in Figure 6.1

Only two layers are enough to pass information between any nodes that are related, because every node in our ADG has distance at most 2 from the original node.

6.2.3 Support Vector Machine

Support vector machine (SVM) [56] is a linear classifier associated with learning algorithm used for binary classification. SVM finds the optimal

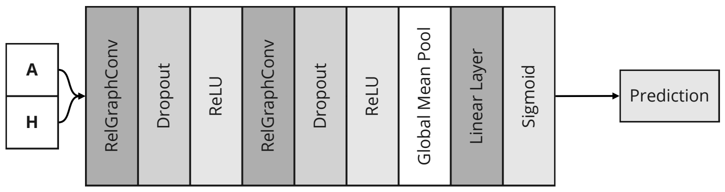

Figure 6.1: The graph neural network uses adjacency matrix A and initial node feature vectors for every node H. Our architecture is two Relation graph convolution layers with ReLU activation function and dropout. Next, we perform global mean pooling over node embeddings, resulting in single vector for entire graph. This vector is than processed using Linear layer and Sigmoid activation function.

separating hyperplane for linearly separable data. This hyperplane maximises the distance, called the margin, from a data sample to the hyperplane. In the case that the data are not linearly separable, soft-margin SVM introduces a trade-off between the size of the margin and penalty of missclassified data samples.

The algorithm also extends to non-linear data using the so-called kernel trick. The core idea of the kernel trick is to create a mapping:

$$\psi : R^d \to \mathbb{R}^e \tag{6.11}$$

usually $e > d$. Although the data $x \in \mathbb{R}^d$ may not be linearly separable, they can be linearly separable as $\psi(x) \in \mathbb{R}^e$. The kernel trick can be extended for non-euclidean data, such as graphs.

There are many graph kernels propsed in the literature [57]. We experiment with three different kernels: Weisfeiler-Lehman kernel [58], Weisfeiler-Lehman Optimal Assignment kernel [59] and Hardman Code kernel [60].

All the kernels use iterative relabeling of nodes. Every node u starts with a label $l^0(u)$ and in every iteration the label is updated as

$$l^{k+1}(u) = f((\bigcup_{v \in N(u)} l^k(v)) \cup l^k(u)) \tag{6.12}$$

where $N(u)$ is the neighbourhood of u and f is a relabeling function, different for each kernel. After fixed amount of iterations, a vector $\phi(G) \in \mathbb{R}^d$ is created by counting the number of occurrences of every label during all the iterations and using L_1 to normalize the vector.

Chapter 7

Experiments

To evaluate our methods, we split the dataset into train and test sets. This split is performed with stratification, depicted in Firure 7.1, a technique for random sampling data with the same proportion of classes to the original set. Stratification is often used when sampling small datasets to avoid unbalanced sampled sets. The dataset is split into train and test sets so that test set is 20% of the entire dataset. Using stratification, the resulting training and validation sets have 196 and 49 entries respectively.

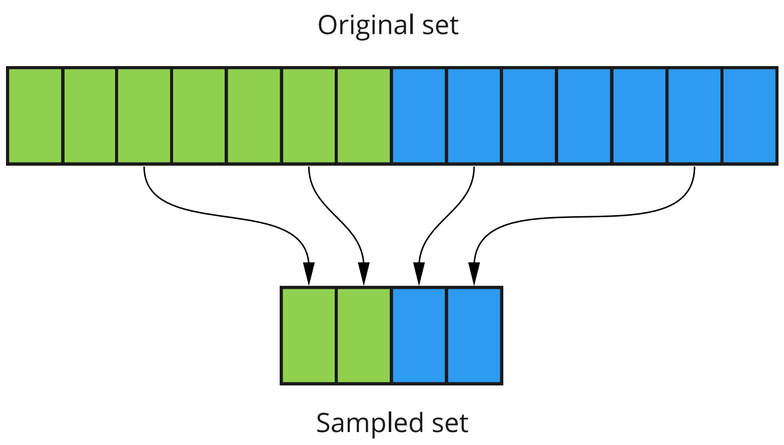

Figure 7.1: Stratified sampling of four samples from the original set. The sampled set is randomly selected from the original set, however, the proportion of classes will always be the same.

Every method requires its own set of hyperparameters. We perform grid search for all hyperparameters with stratified k-fold cross validation of the train set. The k-fold cross-validation divides a set into k groups. A model is trained k times, using $k-1$ groups as a training set and one as a validation set. k-fold cross validation allows us to measure how the model, with its hyperparameters, performs on different data. This method helps us select the hyperparameters, which are then used to train a model on the entire train set. We select $k = 5$ so that the groups are representative.

7.1 Random Forest

Hyperparameters to be chosen are the maximum depth and the number of decision trees in the random forest. The range of parameter search is given in Table 7.1

Parameter	Parameter range
Max decision tree depth	1, 2, 3, 4, 5, 6, 7, 8
Number of decision trees	1, 5, 10, 20, 50, 100

Table 7.1: Hyperparameters range for random forest classifier.

Decision tree trained on k samples cannot have depth more than $\lceil log_2(k) \rceil$, so the maximum depth is upper bounded to 8 when the dataset used has almost 200 samples. The number of decision trees was chosen to increase with larger intervals between the parameters because the expected improvement increases logarithmically, with every additional decision tree adding relatively less.

7.2 Support Vector Machine

We experimented with three different graph kernels to create an embedding of a graph. Weisfeiler-Lehman kernel, Weisfeiler-Lehman optimal assignment (OA) kernel, and Hardman code kernel. These kernels require only the number of iterations as a hyperparameter. The number of iterations defines how many times the relabeling process is performed. SVM uses only C, the regularization term, as a hyperparameter. The range of parameters is shown in Table 7.2

Parameter	Parameter range
Number of iterations	1,2,3,4,5,6,7,8,9,10
C	0.125, 0.25, 0.5, 1, 2, 4, 8

Table 7.2: Hyperparameters range for graph kernel SVM.

7.3 Graph Neural Network

7.3.1 Hyperparameter Selection

We approach training GNN differently from the other methods. This is because neural networks are trained iteratively. To avoid overfitting, we use early stopping. This technique requires measuring the performance of the model on unseen data to evaluate the generalization capability of the model. One option of early stopping is to save models after every learning epoch. When the training finishes, the model that performed best on the unseen data is selected as the final one.

To enable early stopping, we subdivided validation set from the train set. This leaves us with three sets for every cross-validation. For parameter selection, we have two sets: train set, that is used for cross validation) and validation set used to evaluate the generalization capability.

The model is trained using binary cross-entropy loss

$$L = -\frac{1}{n}\sum_{i=1}^{N} y_i\, log(q(y_i)) - (1-y_i) log(q(1-y_i)) \tag{7.1}$$

where y denotes class label (either 1 for propaganda or 0 for non-proapaganda) and q is the model's score of y_i being propaganda. The model is trained using Adam optimizer.

Parameters of this model to be estimated are: hidden layer size, dropout, and learning rate for optimizer. The range of parameters is in Table 7.3.

Parameter	Parameter range
Hidden size	4, 8, 12, 16, 24, 32
Dropout	0, 0.1, 0.3, 0.5, 0.7
Learning rate	0.1, 0.05, 0.01, 0.005, 0.001

Table 7.3: Hyperparameter range for GNN models.

We train the GNN for 100 epochs. After every epoch, we save the model. We selected the best model, out of the 100, based on the accuracy on the validation split of the cross validation. We evaluate the model on the validation set. This is chosen to test the generalization capability of the model. We select the best parameters as the ones that yield the best mean validation accuracy.

7.3.2 Proposed Models

We created 12 models, each using various subsets of features introduced in Section 6.1. Every edge feature denotes its own edge type, therefore, increasing the number of edge features increases the complexity of the model. All 12 models with the features they are using are in Table 7.4.

Models 2,6,10 with five edge features require 6 weight matrices per layer (five for every edge type and one for the self-loop weight matrix, see Equation 6.10). Models with only one edge feature, such as 3 and 4, require only two weight matrices, therefore, decreasing the number of parameters to the more complex ones.

Model	Node features	Edge features
1	V_{SNA}	E_u, E_t, E_d
2	V_{SNA}	$E_u, E_t, E_d, E_{ru}, E_{rt}$
3	V_{SNA}	E_a
4	V_{SNA}	E_{ra}
5	V_{TD}, V_l	E_u, E_t, E_d
6	V_{TD}, V_l	$E_u, E_t, E_d, E_{ru}, E_{rt}$
7	V_{TD}, V_l	E_a
8	V_{TD}, V_l	E_{ra}
9	V_{SNA}, V_{TD}, V_l	E_u, E_t, E_d
10	V_{SNA}, V_{TD}, V_l	$E_u, E_t, E_d, E_{ru}, E_{rt}$
11	V_{SNA}, V_{TD}, V_l	E_a
12	V_{SNA}, V_{TD}, V_l	E_{ra}

Table 7.4: 12 Proposed GNN Models

Chapter 8

Results and Discussion

8.1 Results

In the previous chapter we found the best hyperparameters for our models. Training of the random forest and graph kernel SVM differs to training of a GNN. For random forest and graph kernel SVM, we use entire train set as described in Chapter 7. GNNs requires monitoring during the training, therefore we keep the data split as described in Section 7.3.1.

Random forest with 50 decision trees and maximum depth 5 achieved 81.63 % test accuracy. Result of graph kernel SVM is in Table 8.1. The best kernel for SVM is Weisfeiler-Lehman OA which gives 71.43 % accuracy on test set.

Kernel	**Number of iterations**	**C**	**Test accuracy**
Weisfeiler-Lehman OA	1	8	**71.43** %
Weisfeiler-Lehman	2	2	61.22 %
Hadamard code	4	0.125	57.14 %

Table 8.1: Result of graph kernel SVM

GNN models were trained using the process described in Section 7.3.1. The

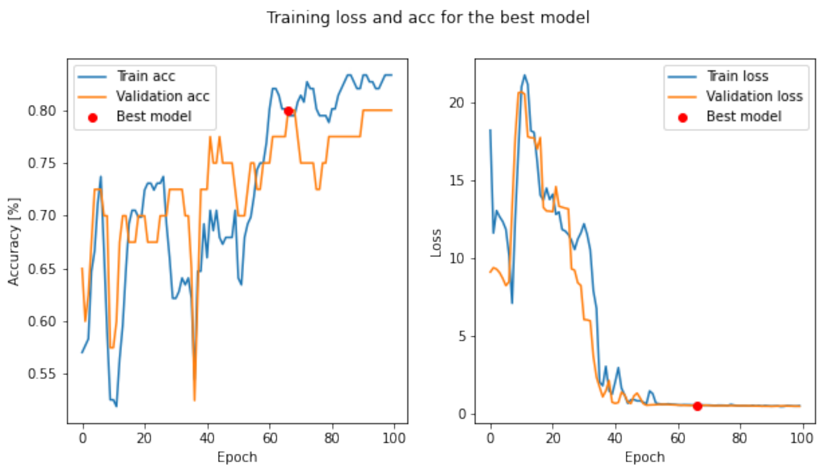

Figure 8.1: Training curves of a GNN model 1, the one using node features V_{SNA} and edge features E_u, E_t, E_d. We select the best model out of the 100 based on the maximum accuracy on the validation set, which is after 69 epochs. This model achieved 75 % accuracy on the test set.

train set was split into training and validation set. We train the model on the training set for 100 epochs and evaluate after every epoch on the validation set. We select the best model from the training process based on the score on the validation set. Learning curves for one parameter selection for the cross validation are seen in Figure 8.2. This best results we could achieved with GNN is 75 % accuracy on the test set. Training of this model as well as the model selection is seen in Figure 8.1.

8.2 Discussion

The random forest was best performing model with over 80 % test accuracy. However, all the models performed well, considering that the data nature is difficult. The best graph neural network model scored 75 % test accuracy. The best models were the ones using social network analysis features. We believe that adding the SNA features to describe the position of nodes in a graph was helpful to our models.

Graph kernel-SVM under-performed compared to the other methods because the method primarily uses the graph structure without additional information, like the article's publication dates and edge types. Moreover, most research focus shifted from graph kernels to GNNs [57]. We included

Model	Hidden size	Dropout	Learning rate	Mean CV accuracy	Test accuracy
1	32	0.5	0.01	**84.63 %**	**75.51 %**
2	32	0	0.01	82.69 %	65.31 %
3	32	0.5	0.01	73.00 %	59.18 %
4	32	0.5	0.001	77.60 %	69.39 %
5	32	0.3	0.001	73.00 %	59.18 %
6	10	0.5	0.01	78.83 %	69.39 %
7	16	0.3	0.001	69.23 %	51.02 %
8	16	0	0.005	73.65 %	65.31 %
9	16	0.3	0.01	75.06 %	63.26 %
10	16	0	0.001	80.74 %	61.22 %
11	16	0.3	0.05	71.77 %	**75.51 %**
12	16	0.3	0.001	74.39 %	73.46 %

Table 8.2: Results of the 12 GNN models propsed in Table 7.4. We found the best parameters using cross validation for every model, and evaluated each model on the test set.

graph kernels in our research because of the limited dataset size.

Deep learning methods are known to require a lot of data. We think the size of our dataset is the reason why random forest outperformed GNNs. For future work, we believe the best approach for classifying articles using the distribution pattern is using graph neural networks.

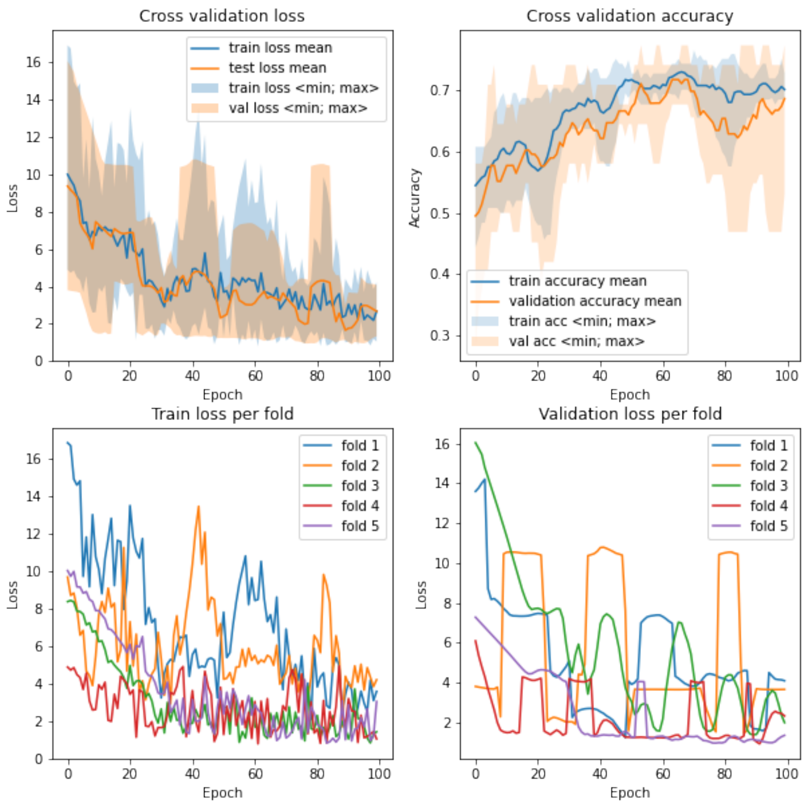

Figure 8.2: Learning curves for cross validation. We train 5 models for every set of parameters using the 5-fold cross validation. We measure the train and validation loss, shown as mean with the range of minimum and maximum losses for every fold.

Chapter 9

Conclusion and Future work

9.1 Conclusion

In this thesis, we developed a novel approach to computational propaganda detection. We evaluated whether it is possible to identify propaganda web articles using the knowledge of how the news is distributed on the Internet. One of the obstacles we faced during the research was the absence of a dataset of propaganda and non-propaganda web articles, which we needed for our work.

As part of this thesis, we developed the CTU-Propaganda-V1 dataset, consisting of propaganda and non-propaganda web articles, their URLs, and other features, such as date of publication, text content, HTML, etc. For propaganda articles, the primary source was the EUvsDisinfo project. Each article in the database was manually examined for the availability and occurrence of automatic re-directions. To collect non-propaganda articles, we manually searched and fact-proved articles with similar topics as the propaganda articles from EUvsDisinfo. Because EUvsDisinfo mostly has articles regarding politics in Russia, so, for non-propaganda news, we stayed in the same social context to decrease the bias of how the distribution on the Internet would look like, and we looked for truthful news in Russian media sources. Because of the language barrier, we have to rely on Google Translate and a native Russian speaker to translate each article and then check if it corresponds to the topics pre-chosen in propaganda articles. The process was carried out manually and was time-consuming.

As a second contribution to the thesis, we developed the Werge tool, which searches the Internet using either the title of the article or its URL, in various search engines and social networks, and then builds a *Article Distribution Graph* based on the results found. An Article Distribution Graph is the graph of all the web articles linking back to the main article. We designed and trained an approach to extract the article's date of publication from the HTML and to perform text similarity between two articles.

Moreover, we designed three models for graph classification tasks, exploring a different approach. The Random Forest uses features extracted from a distribution graph; however, it does not use the graph structure. Graph kernels use very little information about the features (only the level) and focus purely on the graph structure. Graph neural network uses both as graph structure, so feature extraction. We believe that machine learning approach is a promising way to detect propaganda articles based on their ADG. The best result with 81.63 % accuracy was found with Random Forest classifier.

The main contributions of this work include:

- We created CTU-Propaganda-V1 dataset containing 245 articles as well as their ADGs.
- We developed a method to create an Article Distribution Graph of an Internet article using Werge.
- We compared different machine learning algorithms for detecting computational propaganda.
- We trained a Random Forest classifier with 81.63 % accuracy on unseen data.
- We show the Internet distribution pattern of an article is sufficient to achieve up to 80% accuracy on unseen articles.

9.2 Future Work

One of the most challenging but promising tasks is creating a benchmark dataset that can compare different approaches in propaganda classification using distribution graphs. There are two possible approaches to how the benchmark could look like: a dataset with propaganda/ non-propaganda URLs and a dataset with propaganda/non-propaganda ADGs. On the one hand, a dataset of URLs can quickly become outdated since the Internet is ever-changing, and the search engines prefer newer results to older ones. On the other hand, creating a dataset of distribution graphs limits the researchers to use of their method for building ADG. A larger dataset also allows using more complex methods.

Another promising research venue could be in the area of data extraction and article comparison. We wanted to avoid more complex NLP methods for comparing articles because, for our research, it was more important to find articles that share parts of the texts. Using NLP methods to compare the meaning of articles rather than exact words might also be an additional area of improvement.

Appendix A

Bibliography

[1] A. Johnson, "Propaganda analysis and public speaking," *The Southern Speech Journal*, vol. 4, no. 3, pp. 12–15, 1939.

[2] S. C. Woolley and P. N. Howard, "Political communication, computational propaganda, and autonomous agents: Introduction," *International journal of Communication*, vol. 10, 2016.

[3] S. C. Woolley and P. Howard, "Computational propaganda worldwide: Executive summary," 2017.

[4] E. Ferrara, O. Varol, C. Davis, F. Menczer, and A. Flammini, "The rise of social bots," *Communications of the ACM*, vol. 59, no. 7, pp. 96–104, 2016.

[5] Z. Yang, Z. Hu, C. Dyer, E. P. Xing, and T. Berg-Kirkpatrick, "Unsupervised text style transfer using language models as discriminators," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[6] . H. Rashkin, E. Choi, J. Y. Jang, S. Volkova, and Y. Choi, "Truth of varying shades: Analyzing language in fake news and political fact-checking," pp. 2931–2937, 2017.

[7] A. Barrón-Cedeno, I. Jaradat, G. Da San Martino, and P. Nakov, "Proppy: Organizing the news based on their propagandistic content," *Information Processing & Management*, vol. 56, no. 5, pp. 1849–1864, 2019.

[8] V. Baisa, O. Herman, and A. Horak, “Benchmark dataset for propaganda detection in Czech newspaper texts,” in *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2019)*, (Varna, Bulgaria), pp. 77–83, INCOMA Ltd., Sept. 2019.

[9] J. Zhang, R. Zhang, Y. Zhang, and G. Yan, “The rise of social botnets: Attacks and countermeasures,” *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 1068–1082, 2016.

[10] “Euvsdisinfo.” https://euvsdisinfo.eu/. Accessed: 2021-08-15.

[11] T. Quandt, L. Frischlich, S. Boberg, and T. Schatto-Eckrodt, “Fake news,” *The international encyclopedia of journalism studies*, pp. 1–6, 2019.

[12] D. M. Lazer, M. A. Baum, Y. Benkler, A. J. Berinsky, K. M. Greenhill, F. Menczer, M. J. Metzger, B. Nyhan, G. Pennycook, D. Rothschild, *et al.*, “The science of fake news,” *Science*, vol. 359, no. 6380, pp. 1094–1096, 2018.

[13] “Fake news,” in *Cambridge dictionary*, https://dictionary.cambridge.org/dictionary/english/fake-news. Accessed 5/5/2022.

[14] ““misinformation” vs. “disinformation”: Get informed on the difference.” https://www.dictionary.com/e/misinformation-vs-disinformation-get-informed-on-the-difference/, May 2020. Accessed: 5/5/2022.

[15] “Propaganda,” in *Cambridge dictionary*, https://dictionary.cambridge.org/dictionary/english/propaganda. Accessed 5/5/2022.

[16] J. Fawkes and K. Moloney, “Does the european union (eu) need a propaganda watchdog like the us institute of propaganda analysis to strengthen its democratic civil society and free markets?,” *Public Relations Review*, vol. 34, no. 3, pp. 207–214, 2008.

[17] “Propaganda techniques,” https://codlrc.org/evaluating/propagandaa. Accessed 5/5/2022.

[18] B. C. Boatwright, D. L. Linvill, and P. L. Warren, “Troll factories: The internet research agency and state-sponsored agenda building,” *Resource Centre on Media Freedom in Europe*, vol. 29, 2018.

[19] C. Grimme, M. Preuss, L. Adam, and H. Trautmann, “Social bots: Human-like by means of human control?,” *Big data*, vol. 5, no. 4, pp. 279–293, 2017.

[20] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[21] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, *et al.*, "Palm: Scaling language modeling with pathways," *arXiv preprint arXiv:2204.02311*, 2022.

[22] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[23] B. Sanchez-Lengeling, E. Reif, A. Pearce, and A. B. Wiltschko, "A gentle introduction to graph neural networks," *Distill*, 2021. https://distill.pub/2021/gnn-intro.

[24] W. Y. Wang, "" liar, liar pants on fire": A new benchmark dataset for fake news detection," *arXiv preprint arXiv:1705.00648*, 2017.

[25] G. Da San Martino, S. Yu, A. Barrón-Cedeno, R. Petrov, and P. Nakov, "Fine-grained analysis of propaganda in news article," in *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 5636–5646, 2019.

[26] G. Da San Martino, A. Barrón-Cedeno, H. Wachsmuth, R. Petrov, and P. Nakov, "Semeval-2020 task 11: Detection of propaganda techniques in news articles," in *Proceedings of the fourteenth workshop on semantic evaluation*, pp. 1377–1414, 2020.

[27] "Lexical and computational semantics and semantic evaluation." https://aclanthology.org/venues/semeval/. Accessed: 5/5/2022.

[28] T. Mitra and E. Gilbert, "Credbank: A large-scale social media corpus with associated credibility annotations," in *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 9, pp. 258–267, 2015.

[29] E. Tacchini, G. Ballarin, M. L. Della Vedova, S. Moret, and L. de Alfaro, "Some like it hoax: Automated fake news detection in social networks," *arXiv preprint arXiv:1704.07506*, 2017.

[30] D. B. Buller and J. K. Burgoon, "Interpersonal deception theory," *Communication theory*, vol. 6, no. 3, pp. 203–242, 1996.

[31] M. Ott, Y. Choi, C. Cardie, and J. T. Hancock, "Finding deceptive opinion spam by any stretch of the imagination," *arXiv preprint arXiv:1107.4557*, 2011.

[32] J. W. Pennebaker, R. L. Boyd, K. Jordan, and K. Blackburn, "The development and psychometric properties of liwc2015," tech. rep., 2015.

[33] R. Mihalcea and C. Strapparava, "The lie detector: Explorations in the automatic recognition of deceptive language," in *Proceedings of the ACL-IJCNLP 2009 conference short papers*, pp. 309–312, 2009.

[34] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[35] S. Yoosuf and Y. Yang, "Fine-grained propaganda detection with fine-tuned bert," in *Proceedings of the second workshop on natural language processing for internet freedom: censorship, disinformation, and propaganda*, pp. 87–91, 2019.

[36] T. Alhindi, J. Pfeiffer, and S. Muresan, "Fine-tuned neural models for propaganda detection at the sentence and fragment levels," *arXiv preprint arXiv:1910.09702*, 2019.

[37] P. Gupta, K. Saxena, U. Yaseen, T. Runkler, and H. Schütze, "Neural architectures for fine-grained propaganda detection in news," *arXiv preprint arXiv:1909.06162*, 2019.

[38] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *arXiv preprint arXiv:1508.01991*, 2015.

[39] G. D. S. Martino, S. Cresci, A. Barrón-Cedeno, S. Yu, R. Di Pietro, and P. Nakov, "A survey on computational propaganda detection," *arXiv preprint arXiv:2007.08024*, 2020.

[40] A. Bessi and E. Ferrara, "Social bots distort the 2016 us presidential election online discussion," *First monday*, vol. 21, no. 11-7, 2016.

[41] A. M. Kaplan and M. Haenlein, "Users of the world, unite! the challenges and opportunities of social media," *Business horizons*, vol. 53, no. 1, pp. 59–68, 2010.

[42] Y. Boshmaf, I. Muslukhov, K. Beznosov, and M. Ripeanu, "Design and analysis of a social botnet," *Computer Networks*, vol. 57, no. 2, pp. 556–578, 2013.

[43] Q. Cao, M. Sirivianos, X. Yang, and T. Pregueiro, “Aiding the detection of fake accounts in large scale social online services,” in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pp. 197–210, 2012.

[44] Z. Yang, C. Wilson, X. Wang, T. Gao, B. Y. Zhao, and Y. Dai, “Uncovering social network sybils in the wild,” *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 8, no. 1, pp. 1–29, 2014.

[45] S. Kudugunta and E. Ferrara, “Deep neural networks for bot detection,” *Information Sciences*, vol. 467, pp. 312–322, 2018.

[46] B. Wu, L. Liu, Y. Yang, K. Zheng, and X. Wang, “Using improved conditional generative adversarial networks to detect social bots on twitter,” *IEEE Access*, vol. 8, pp. 36664–36680, 2020.

[47] V. S. Subrahmanian, A. Azaria, S. Durst, V. Kagan, A. Galstyan, K. Lerman, L. Zhu, E. Ferrara, A. Flammini, and F. Menczer, “The darpa twitter bot challenge,” *Computer*, vol. 49, no. 6, pp. 38–46, 2016.

[48] J. Echeverrï£¡ a, E. De Cristofaro, N. Kourtellis, I. Leontiadis, G. Stringhini, and S. Zhou, “Lobo: Evaluation of generalization deficiencies in twitter bot classifiers,” in *Proceedings of the 34th Annual Computer Security Applications Conference*, pp. 137–146, 2018.

[49] B. Hooi, N. Shah, A. Beutel, S. Günnemann, L. Akoglu, M. Kumar, D. Makhija, and C. Faloutsos, “Birdnest: Bayesian inference for ratings-fraud detection,” in *Proceedings of the 2016 SIAM International Conference on Data Mining*, pp. 495–503, SIAM, 2016.

[50] A. Chetan, B. Joshi, H. S. Dutta, and T. Chakraborty, “Corerank: Ranking to detect users involved in blackmarket-based collusive retweeting activities,” in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, pp. 330–338, 2019.

[51] S. Garcia and E. Babayeva, “Werge: Web Reverse Graph Extractor,” 8 2021.

[52] L. SerpApi, “Serpapi.”

[53] Crummy, “Beautifulsoup4.”

[54] A. Barbaresi, “htmldate: A Python package to extract publication dates from web pages,” *Journal of Open Source Software*, vol. 5, no. 51, p. 2439, 2020.

[55] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," 2017.

[56] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[57] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *Applied Network Science*, vol. 5, no. 1, pp. 1–42, 2020.

[58] N. Shervashidze, P. Schweitzer, E. J. Van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels.," *Journal of Machine Learning Research*, vol. 12, no. 9, 2011.

[59] N. M. Kriege, P.-L. Giscard, and R. Wilson, "On valid optimal assignment kernels and applications to graph classification," *Advances in neural information processing systems*, vol. 29, 2016.

[60] T. Kataoka and A. Inokuchi, "Hadamard code graph kernels for classifying graphs.," in *ICPRAM*, pp. 24–32, 2016.