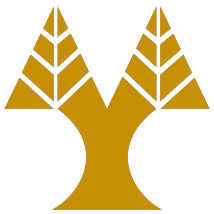
Thesis Dissertation

FAKE NEWS: DETECTION

Ioannis Sophocleous

UNIVERSITY OF CYPRUS



DEPARTMENT OF COMPUTER SCIENCE

January 2021

UNIVERSITY OF CYPRUS

DEPARTMENT OF COMPUTER SCIENCE

Fake News: Detection

Ioannis Sophocleous

Supervisor

Dr. George Pallis

Thesis submitted in partial fulfilment of the requirements for the award of degree of Bachelor in Computer Science at University of Cyprus

January 2021

Acknowledgments

I would like to thank the following people, without whom I would not have been able to complete this thesis, and without whom I would not have made it through my bachelor's degree!

My supervisor Dr George Pallis, Associate Professor in the Computer Science Department of the University of Cyprus, who using his great enthusiasm of this field urged me to explore one of the most significant and most important issues of the new age of information, which has challenged me in every way and in doing so has made me stronger and better.

Next, I would like to thank the MSc candidate Chrysovalantis Christodoulou for his excellent guidance, endless support and patience throughout my research and the chance he gave me to learn an incredible amount of technologies.

Everyone who helped me with the crowd-sourcing of thousands upon thousands of documents for the manual evaluation of my research. You are real MVP's here.

Finally, my biggest thanks go to my family for all the unconditional support in this very intense academic year. To my parents George and Yianna - I promise to spend less time in front of the computer screen from now on!

Abstract

Nowadays, the presence of fake news on the Internet is a common and daily phenomenon. The freedom of speech and expression that characterises the Internet has become a platform of exploitation and misinformation to serve interests that spread confusion and has made the uncontrolled rise of fake news an important social issue. In order to find new ways to reduce the spread of this infection, it is first necessary to understand its behaviour and how it spreads. Our thesis has multiple primary goals. First, we contribute to a greater project for "Fake News Evolution", whose aim is to examine the evolution of fake news by gathering a vast amount of articles from 2009 to 2019. Furthermore, to help people identify fake news over the web, we experimented with various state-of-the-art algorithms for detecting near-duplicate articles, web-scraping and natural language processing techniques. Using these methods, we have developed a system that, upon given the URL of a suspicious article, searches across various well-acclaimed fact-checking domains to prove the articles' misinformation. With our vast collection of fake news articles, we evaluated our system's performance on 16.5k manually labelled articles. We have also provided a highly programmable annotation platform that can be easily modified for future experimentation to further optimise our system.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Motivation

Since the beginning of the 20th-century journalists have dealt with the spread of misinformation without many obstacles. However, with the rise of information and communication technologies and its immense impact on our society, fake news became an everyday phenomenon. The rapid growth of the Internet and the enormous influence of the Internet media in our lives makes the problem of the uncontrolled rise of fake news an important social issue. For the above reasons, the necessity for the reinvention of how journalists work but also how the public is informed has sparked the interest of many researchers and the general public. This interest began escalating with the 2016 United States elections [1] and continued to expand with the U.K. referendum. On even more recent events, the Covid-19 virus and the 2020 United States elections have also been a cause of a new outbreak for fake news misinformation. In particular, rumours and false information for Covid-19 circulating social media became so increasingly challenging to distinguish from the real facts that many governments and authorities urged citizens to confirm a news stories' veracity before circulating them.

In addition, the fact that the president of one of the most powerful nations in the world is affected by fake news dissemination has baffled the public and forced them to awake to the idea of misinformation. Figure 1.1 shows the increase of interest over time with the search for fake news in the Google search engine. It clearly shows this exponential increase in public's interest in the term "fake news" by the end of 2016 and forward. A few significant peaks on the graph were in February of 2017 when President Trump used the phrase "fake news" in several of his tweets and interviews. Furthermore, on the 14th of January of 2018, he attacked the Wall Street Journal as 'fake news' over his North Korea comments, and lastly on March of 2020 when Covid-19 related fake news escalated as the virus took a global scale.

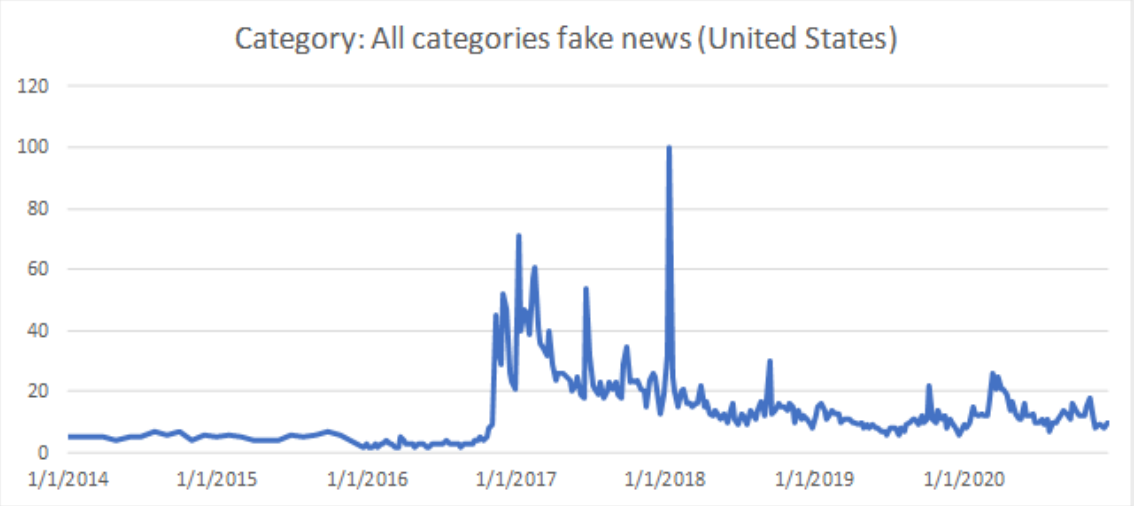


Figure 1.1: Google trends for U.S. in category: Fake News

Even though the need for an autonomous fake news detection system is undeniable, not much work has been done to cover the demand due to the vast complexity of the problem. Humans find it very difficult to distinguish between real and fake news without extensive knowledge of the topic in question. As Automated Fact-Checking concerns many different academic fields, it is expected to be an inconsistency in the terminology used across them. With this survey, Thorne and Vlachos [18] identify the issue and try to unify the task formulations and methodologies for better future understanding and research across those fields. Furthermore, Pérez-Rosa et al. [12] describe the process of collection, annotation, and validation of fake news in more detail, while showing the different approaches

used in their detection. In this research, we are focusing on creating a fake news detection system using fact-checking domains as a reference to determine the classification of the article.

1.2 Challenges

Overall, Fake News detection is a very complex and challenging problem. Even fact-checking experts, like journalists, need to conduct thorough research for the topic of every news article they want to check to give a correct confirmation of the articles' claim. Furthermore, the binary classification of truth or fake is not always possible as it might be partial truth or false. An even more challenging case is if the article is intentionally exaggerated to give a satirical view of a subject for the readers' entertainment. The identification process becomes even more challenging when expert journalists write the article with the malicious goal of disinforming the reader to comply with their political agenda, as we have seen with the 2016 United States elections, the U.K. referendum for leaving the U.K. and the Covid-19 outbreak. These journalists are experts in fabricating the truth in a well-written article while hiding their purpose behind strong and powerful words, making it harder to extract the article's exact claim.

The Internet, being the most extensive collection of data globally, makes the process of identifying similar articles like searching for a needle in a haystack. Therefore, we need to filter our search to the best possible way. In doing so, the claim of the article needs to be extracted. Finding a text article's claim is an open problem in the world of natural language processing, as shown by already conducted research [7], as text is complicated for a computer to understand its meaning entirely. Thus the outcome might not be as accurate as expected.

In addition, since the fact-checking domains make a complete analysis of any questionable article by trying to explain its falsy claim using facts from different trusted sources, another problem we need to face is to correctly identify that the article we are trying to disprove is the same, or similar, to the article we are trying to classify. Using techniques for near-duplication text and natural language processing, we need to identify the rela-

tionship between the two; thus we need to be very precise in our metrics and the variables we use in order to get the best result.

The data collection is another challenging part of the research. Most of the manually labelled datasets are too small, making them inadequate for machine learning approaches. On the other hand, the datasets that consist of enough data for such approaches make assumptions to declare the veracity of an article. For example, they declare an article as fake if its origin is an untrusted domain that usually publishes fake news. To deal with this assumption, we need to find a way to automate verifying an article's veracity.

Furthermore, we need to consider the amount of irrelevant information that those datasets might contain, for example all non-English articles, since our algorithms only work for the Latin alphabet. Furthermore, because of the extensive amount of time and resources needed to gather and evaluate the system results, there is not enough time for a thorough investigation and re-examination of the system and its parameters.

Additional to the above challenges, we also had to deal with some technical restrictions. Along with the system's implementation, we also wanted to create an annotation website to host the system and be used as a tool for crowdsourcing the evaluation system's results. Since the system itself needs to perform complex calculations with features such as N-grams and keyword extraction, we required a programming language suitable for that purpose. With a vast collection of libraries in Natural Language Processing and more, the Python language was a perfect candidate. The website, which was implemented as a NodeJS[1] project with the ReactJS[2] library's use, had to communicate with the system written in Python and collect the results in Javascript. Also, the system needed to remain under the University of Cyprus domain to communicate with the database for the crowdsourcing evaluation process and stay secure under the university's firewalls. An API was created using the Flask API[3], which is also written in Python and can run the system in the background. The annotation website communicates with the API using restricted and

[1] `https://nodejs.org/en/`

[2] `https://reactjs.org/`

[3] `https://flask.palletsprojects.com/en/1.1.x/`

secure HTTP requests. While this makes the system slower, it keeps it protected as it communicates with the API.

1.3 Contributions

This research has multiple goals. To better understand the impact of fake news and gain insight into how they continue existing, we conducted an evaluation of their evolution of the past decade. Using a vast dataset with news articles, both fake and truthful, we extracted all relevant information to give a picture of how they behaved over some significant events in the past decade and suggest ways of fighting against misinformation.

Furthermore, another goal is to create a system that, with decent precision, can classify a given article as truthful or fake news by cross-checking with articles from fact-checking domains with similar claim. We use multiple state-of-the-art algorithms for near-duplicate detection, keyword, and claim extraction with web crawlers' help to locate and identify said articles under some of the most well-acclaimed fact-checking domains.

The ultimate goal of our research is to set the roots for the perfection of the above system so future researchers can use it to perform the evaluation and calculate the precision of their fact-checking systems to further our efforts to reduce the spread of fake news and misinformation across the Internet and its users. The general public could also benefit from such a system by fact-checking an article, when they doubt its claim, without the need for extensive knowledge.

Moreover, this research contributes to a study performed in the University of Cyprus for "Fake News Evolution", which developed the web crawlers and gathered articles to complete the dataset collection we are going to use for our needs. Their goal by collecting articles from both trusted and untrusted sources over a period of ten years is to extract different linguistic features from those articles and visualise them to study their evolution over time.

To sum up, our contribution is as follow:

- Implementation of a text similarity system for identifying articles' truthfulness, by utilising known fact-checking organisations.
- Implementation of a user-friendly annotation platform.
- Contribute to a greater project "Fake News Evolution", which gathers a vast amount of articles from 2009 to 2019 to examine the evolution of Fake News.

Our contribution to detecting online misinformation has to be effective and help the researchers and the public to address this problematic issue.

1.4 Outline Contents

The organisation of our contents is as follows.

Chapter 1: Introduction
The Introduction explains our research's motivation and why this is a significant problem that needs to be addressed. It lists the multiple challenges we had to face and the goals we want to achieve to provide a good contribution to our study and the general public.

Chapter 2: Related Work
This chapter focuses on the analytical review of the literature used in creating this paper, consisting of fake news classification and detection techniques with the help of natural language processing. Furthermore, methods for web scraping and detection of near-duplicate articles are listed, with state-of-the-art algorithms used for the implementation of our system.

Chapter 3: Fake News Evolution
In this chapter, we explain in detail the process we took in analysing the evolution of fake news over the past decade. We show the dataset used to collect the data, the process of feature selection, and the cleaning of unwanted data. Then we present the visual representation and give a description of our findings.

Chapter 4: Implementation: Fake News Identifier

Here we define the methodology and explain each step of the implementation of the system for fake news identification. We explain the reasoning behind each variable used for the training of the algorithms and how they work. Then we give a detailed description of the annotation system, how it operates, and how it is implemented with the best user experience. Lastly, we give an overview of how crowdsourcing was conducted to gather and evaluate the final results of the system.

Chapter 5: Evaluation

In the evaluation chapter, we describe the datasets we used and their role in our study. Then we conduct an in-depth investigation of the above, presenting our experiments' results and how they measure our original goals, while also giving a detailed description of what did not work and suggestions for future optimisation. We offer a visual comparison of our results and a table with specific values used for each experiment.

Chapter 6: Conclusion

Chapter 6 defines our closing thoughts and conclusions to summarise our research and results. Finally, we provide important future work that could be done to further extend and improve our work and help the research with fake news detection.

Chapter 2

Related Work

Contents

2.1 Fake News Detection

Since the 2016 U.S. presidential elections, where fake news is claimed to have had a significant impact on the result [1], fake news detection has earned much interest in not only Computer Scientists but in journalists as well. Fact-checking as a whole is considered as an arduous process even for a human and even experts claim that someone cannot give an outright opinion of the authenticity of an articles' claim without having extensive knowledge on the whole topic. Furthermore, it is assumed that experts write these articles with the malicious attempt to disinform the reader by writing it to look as realistic as possible at the point where the reader may not even think to question its reliability nor check for its authenticity using sources or evidence in its content. Horne et al. [6] in their study show that fake news is more similar to satire than real news leading us to conclude that persuasion in fake news is achieved through heuristics rather than the strength of arguments. With the use of three different datasets with real, fake and satire political news, gathered from well-known domains and different features that can be categorised as stylistic, complexity and psychological, they prove how title structure and the use of NNP (Proper Noun) are the main threat in persuading the reader about an articles' reliability.

Because declaring an article as real or fake news is somewhat tricky, there is no universal approach to a solution. So it comes to no surprise that there are many different approaches to reach a result. One of these approaches is proposed in the article Five Shades of Untruth: Finer-Grained Classification of Fake News [19] as a classification to distinguish between hoaxes, irony and propaganda by combining the work of other researchers. Wang et al. wanted to show that everything is not just black and white in the classification of fake news, like prior work that focused on binary classification (factual or fake). This approach combined both the SHPT scheme (satire, hoax, propaganda and trusted news) and the Politifact six-way "Truth-O-Meter" (true, mostly true, half true, mostly false, false and pants-on-fire) in a classification hierarchy. By mapping both of their labels into a tree format they were able to capture the five major categories of fakeness: Factual, Hoax, Irony, Incomplete Propaganda and Manipulative Propaganda, and in doing so, give a more in-depth understanding in the pattern of fake contents and how it influences and spreads to the public. The results also showed that combining content and social media information can improve the prediction quality.

The spread of fake news does not stop on blogs and news articles. Social media sites have also been known to spread misinformation uncontrollably. More approaches have been developed in analysing the network characteristics and the spread of fake news over social media to extract features that will help us determine an article's truthfulness. To further understand this phenomenon, the scientific study for creating the FakeNewsNet dataset [15] has analysed data collected in the Twitter social media platform. By analysing the data, we can see that newly created accounts tend to spread more fake news than others. This is due to social bots creating Sybil attacks on these platforms, which means that a large number of pseudonymous identities are used to gain a disproportionately large influence. Fake news Tweets tend to have a higher ratio of negative sentiment, fewer replies and more retweets, while real news Tweets have more likes, possibly because people agree on them. Also, fake news Tweets are mostly posted at night when most people are inactive, but their density is close to peak hours, possibly due to bot accounts. Although fake news in social media is significantly faster, their life span is shorter and are easier to diffuse due to the revelation of the truthfulness.

2.2 Natural Language Processing

Natural Language Processing, or NLP for short, is a subfield of linguistics, computer science, and artificial intelligence concerned with translating large amounts of natural language data into a programmer-friendly data structure that correctly captures the meaning of the original text including the contextual nuances of the language within them. Since artificial intelligence plays a significant role in Natural Language Processing, machine-learning approaches are the dominant method in analysing text, using SVM (Support Vector Machine) or logistic regression.

In their survey about fake news detection, Conroy et al. [4] introduce several assessment methods for detecting fake news, with one of them being linguistic cue. In linguistic approaches, the content of a message is analysed in search of language patterns that show deception. Data is represented in "n-gram" frequencies. Deep syntax can be implemented using PCFG (Probabilistic context-free grammar) to better predict deception by understanding the syntax used in the document in question. Semantic analysis can be used to understand the meaning behind the linguistic input. Rhetorical Structured and Discourse Analysis identify instances of rhetoric relations between the linguistic elements. They show how the training of classifiers, like in Support Vector Machines (SVM), can be useful in automated numerical analysis.

A similar technique was used in the paper Automatic Detection of Fake News [12] with a focus was on serious news fabrication and celebrity gossip. They used linguistic clues, such as "n-grams", punctuation, psycholinguistic features, readability and syntax, to distinguish between the truth-tellers from liars. The two new datasets they produced comply with the nine requirements of a fake news corpus proposed by Rubin et al. [13]. Several experiments were made using a linear SVM classifier and five-fold cross-validation, with accuracy, precision, recall and F1 measures averaged over the five iterations. Different amounts of training data were designated to find the learning curve and found out that larger training data can improve the classification performance. Another test was performed to assess the human performance of fake news detection in the datasets created. The results showed that humans are better at identifying celebrity fake news than any

other domains. The system created here outperforms humans while detecting fake news in more serious and diverse news sources.

Neural networks are most commonly used in analysing visual images but have shown encouraging results in natural language processing as well. Due to their excellent efficiency in analysing and locating patterns, they can be used to find an article's claim by determining specific sentences or phrases. A great example of a neural network in action is the browser plugin Check-It, developed by faculty and students of the University of Cyprus in collaboration with the University of Crete [11]. This plugin was created to detect fake news over the web and combine different approaches to solve the problem, such as fact-checking, linguistic, social networks and flag-listing, and it is GDPR compliant. It also uses a linguistic model which incorporates textual features (stylistic, complexity, psychological) for headlines and bodies of articles using a trained Deep Neural Network to predict veracity. Due to its top-notch Deep Neural Network, Check-It outperformed other state-of-the-art models on commonly used datasets.

Thorne and Vlachos [18] explained some different types of inputs we can use when using NLP. One frequent input method is subject-predicate-object triples, e.g. (Bob, knows, John), which enables knowledge to be represented in a machine-readable way. Another type of input is textual claims. They are usually short sentences constructed from longer passages where different claims can be considered like numerical claims, entity and event properties, position statements and quote verification. In addition, they declare the difficulties of each approach, and they propose future NLP research on automated fact-checking.

While NLP is not the direct focus of this research, it is essential to understand as it plays a significant impact on the declaration of fake news articles.

2.3 Web Scraping

Web Scraping is the process of using bots to extract content and data from a website. Unlike screen scraping, which copies only the pixels displayed on the screen, web scraping extracts underlying HTML code and replicates entire website content elsewhere. In our research, web scraping is used to extract the vital information of an article, like its title, content, links, and so forth, so we can compare it and check its truthfulness.

Different kind of bots or API can be used to achieve this. One of these uses Scrapy[1], a free and open-source web crawling and web scraping framework used to crawl websites and extract structured data from their pages. A spider bot can easily be scripted to only extract the relevant information of a website. Its advanced item pipelines allow the programmer to write functions in their spider to process their data such as validating, removing or saving their data to a database. Scrapy also provides techniques for preventing the bot from getting blocked by more clever websites by mimicking human-like behaviour, such as the use of random intervals between requests, autothrottle for automatically throttling crawling speed, providing a USER_AGENT for correct identification, and more.

Extracting website information can also be done with more straightforward API like Beautiful Soup[2], parsing the HTML code into a tree that you can easily navigate, search and modify. More advanced options include the Newspaper3k[3] API explicitly designed to extract and curate online articles. Not only does it extract the articles important information like title, authors and text, but it uses Natural Language Processing to extract its keywords and summary as well. Google created its own tool, called Fact Check Explorer[4] [5], that allows users to easily browse and search for fact checks on various fact-checking domains. An example of web crawlers used for information extraction from online articles is done in the survey by Shu et al., who provide a fake news data repository, called FakeNewsNet [15]. The data repository contains two comprehensive datasets that include

[1] `https://docs.scrapy.org/en/latest/intro/tutorial.html`

[2] `https://www.crummy.com/software/BeautifulSoup`

[3] `https://newspaper.readthedocs.io/en/latest`

[4] `https://toolbox.google.com/factcheck/explorer`

[5] `https://developers.google.com/fact-check/tools/api`

news content, social context and dynamic information from different types of news domains such as political and entertainment sources. For the construction of the dataset, different types of crawlers were used. Fact-checking websites were utilised to collect reliable sources for true and fake news, such as PolitiFact and GossipCop. A PolitiFact crawler was used to gather URLs of truthful and fake articles. When the link was deprecated, a Google web search was conducted to identify a closely related article. Another GossipCop crawler was created to gather URLs for entertainment articles. Since most of them are fake stories, true stories were collected from E! Online. A Google search of each article's headline was used to gather the URLs, since GossipCop does not explicitly provide it, using NLP and negative sentiment lexicons to formulate the search query. A news content crawler extracted the original news source from the provided URLs, such as headline, body text, images, author information and links.

2.4 Near Duplicate Detection

When a fact-check is conducted for a particular article, the fact-checking websites either presents a rating based on the accuracy of the article as a whole or they rate each fact individually. The issue here is that different fake news sites can copy a story and provide it with similar content. While one article can be detected as false, several near-duplicates might still be out there. That is why near duplication detection is also an essential field in fake news detection.

An interesting approach for determining the syntactic similarity of documents was done in the paper Syntactic Clustering of the Web by Broder et al. [2]. This method can discover if two documents are "roughly the same" (resemblance) and/or "roughly contained" (containment) in one another. The resemblance and containment are valued from 0 to 1, where each document D can be viewed as a sequence of words, defined as a w-shingle of size w, as S(D, w). So they define the resemblance of two docements A and B as $r(A,B) = |S(A) \cap S(B)| / |S(A) \cup S(B)|$, the containment between them as $c(A,B) = |S(A) \cap S(B)| / |S(A)|$, and the resemblance distance as $d(A,B) = 1 - r(A,B)$. Hence r(A,A) = 1, because the documents resembles itself 100% and the containment if $A \subseteq B$ is c(A,B) = 1. For a shingle of size w, U as the set of all shingles of size w and a fixed param-

eter s where $W \subseteq U$ they define $MIN_s(W)$ as the set of the smallest s elements in W and $MOD_m(W)$ as the set of elements of W that are 0 mod s. Using a random permutation of $\pi : U \rightarrow U$ they define $F(A) = MIN_s(\pi(S(A)))$ and $V(A) = MOD_m(\pi(S(A)))$. So the resemblance is estimated as $|MIN_s(F(A) \cup F(B)) \cap F(A) \cap F(B)| \,/\, |MIN_s(F(A) \cup F(B))|$ or $|V(A) \cap V(B)| \,/\, |V(A) \cup V(B)|$ and the estimated containment as $|V(A) \cap V(B)| \,/\, |V(A)|$. Then in their system, using 10-shingles, with 40-bit Rabin fingerprints as a random permutation, and the "modus" method described above with an m of 25, they generate a list of document pairs that share any shingles, along with the number of their common shingles, and decide if it exceeds the threshold of resemblance to classify them as near duplicate.

Charikar proposes another state-of-the-art method in his research of Similarity Estimation Techniques from Rounding Algorithms [3]. He shows an interesting relationship between rounding algorithms and the construction of locality-sensitive hash functions for classes of objects and shows new construction possibilities. A locality-sensitive hash scheme refers to a family of hash functions F with a collection of objects so that for objects x and y $Pr[h \varepsilon F][h(x) = h(y)] = sim(x,y) \varepsilon [0,1]$. The similarity of objects can be estimated from their compact sketches (signatures) with the use of min-wise independent permutations and the similarity measure of $sim(A,B) = |A \cap B| \,/\, |A \cup B|$ (Jaccard coefficient). Charikar constructed new locality-sensitive hash schemes for a collection of vectors using cosine similarity $[\theta(u,v)/\pi]$ and a collection of distributions on n points in a metric space using the Earth Mover Distance (EMD). The advantage of a locality sensitive hashing based scheme is that this directly yields techniques for the nearest neighbour search for the cosine similarity measure. For the existence of locality-sensitive hash function families, it must satisfy the triangle inequality, maps objects to {0, 1} and corresponds to the similarity function $1 + sim(x,y)/2$ and the distance function $1 - sim(x,y)$ must be isometrically embeddable in the Hamming cube.

Finally, an approach by Henzinger [5] compares the two algorithms discussed above and implements a combined version to get a better result, using the map-reduce framework. For the Broder et al. algorithm [2], pages were converted into token sequences of w-shingles and then hashed into m 64-bit Rabin fingerprints, using m different fingerprint functions. The min value is stored to create an m-dimensional vector of min-values for

each function, which is then reduced to an m'-dimensional vector of supershingles to save space and calculation time. Two pages are near-duplicate iff their supershingle vector similarity is at least 2, called B-similarity. The parameters were set as follows: m = 84, m' = 6 and k = 8. For Charikar's algorithm [3], which is based on random projections of words in documents, each token is projected into a b-dimensional space by randomly choosing b entries from [-1, 1]. All positive entries are set to 1 and all non-positive to 0, resulting in a random projection for each page. Two pages are near-duplicates if the number of agreeing bits in their projections is above a fixed threshold t, called C-similarity. For both algorithms to require the same amount of space per document t was set to 372. Henzinger's combined algorithm first computes all B-similar pairs and then filters out those pairs whose C-similarity falls below a certain threshold. While neither of the two algorithms alone performed well on pages from the same site, this combined algorithm performed well without sacrificing mush recall.

Chapter 3

Fake News Evolution

Contents

3.1 Data Collection

In an effort to understand how fake news have continued to affect us through the years and show how important it is for more research to be conducted in the detection and limitation of fake news over the web, we completed an analysis of their evolution over the last decade. We explain the data used in this analysis, how we prepared it, and how we visualise our findings.

3.1.1 Data Overview

The first step was the data collection and finding the best dataset for our needs. Fortunately, one was provided to us by the University of Cyprus where the "Fake News Corpus"[1] open-source dataset was used and expanded using locally developed web crawlers to gather articles from trusted and untrusted sources, covering a period of ten years from 2009. While the "Fake News Corpus" dataset collected different types of articles from satire, to political, to conspiracy theories, only articles from the Fake News and Credible

[1] `https://github.com/several27/FakeNewsCorpus`

categories were used to create this dataset.

This dataset provides a comprehensive collection of articles containing almost daily basis data with 581.9K documents published from 2009 to 2019. Most of the URL were collected from the web-archive[2], an Internet archive storing past websites, allowing the user to go "back in time" and gather the information that might have been modified or deleted. Table 3.1 explains the columns that make the dataset that was used.

Columns	Description
_id	Unique identifier for each new document
title	The title of the published article
author	The author who wrote the article
publish_date	The timestamp when the article was published
content	The content of the article
domain	The url of the domain that published the article
timestamp	The timestamp the article had when it was extracted
year	The year the article was published
url	The url where the article was published
fake	A boolean to express if the article is considered fake or truthful

Table 3.1: Columns of the new Fake News Corpus dataset

3.1.2 Data Cleaning

After examining the data, it was clear that a subset of the URL in the dataset was either corrupt, deprecated or did not even contain an article. Many advanced regular expressions were implemented to remove the unwanted data to have a more clear final result.

The regular expressions passed from each URL in the original collection. They checked its format for any irregularities like false protocol identifiers (such as HTTP, HTTPS), corrupt web-archive pages since each URL must comply with some guidelines created by

[2] `https://archive.org/about/`

the domain like having a 14 integers long unique key identifier, or if the URL was collected from social media sites (such as Facebook, Twitter, Youtube). If any of the URLs complied with the above, it was considered false data, which this analysis is interested in, and was removed.

The updated dataset holds the new collection of data that was optimised for our needs. This dataset contains 573K entries, meaning that we have cleared about 2GB of unwanted data. The collection is still following the same structure as before.

3.2 Data Visualisation

An understanding of the data was achieved via data visualisation techniques. Data visualisation helps to get a better awareness of the data and extract conclusions from it. We used the Matplotlib plotting library for the Python programming language to produce our visualised data. We produced three types of graph: Bar Chart, Pie Chart and Scatter plots.

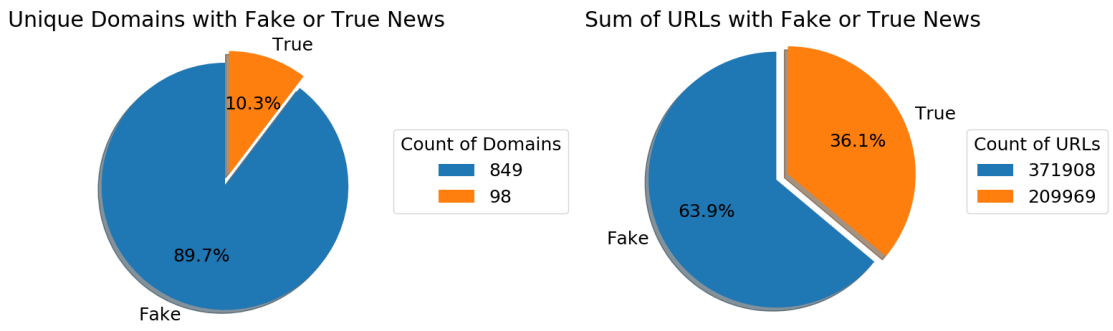


Figure 3.1: Unique domains and URLs classified by their type (fake or true)

Figure 3.1 displays two Pie charts to show the dataset's variety. The left chart shows how many unique domains are included in the dataset. With a total of 947 domains, at the time of writing, a stunning 89.7% are domains that published an article classified as fake news. The right chart, however, displays the total of unique URLs that consist in the dataset. With a 63.9% total of URL that links to fake news articles, it shows that the dataset is well balanced with both true and fake articles.

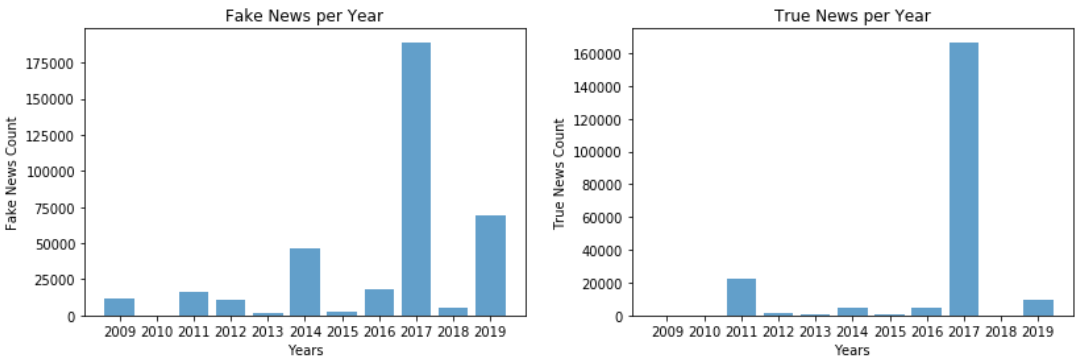
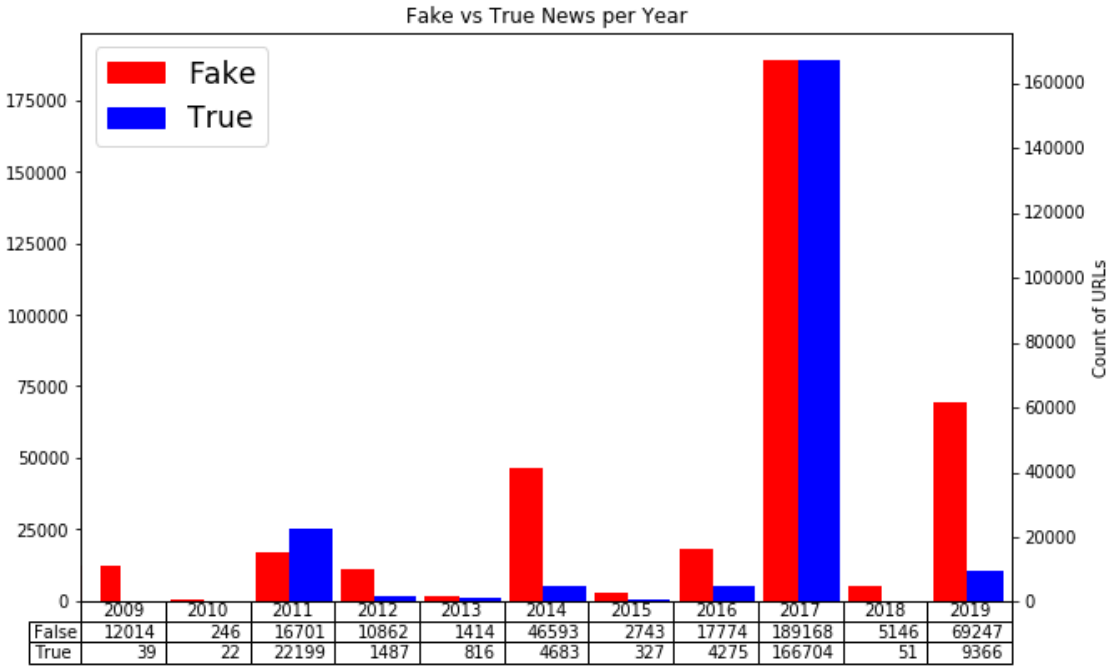


Figure 3.2: Fake and True articles classified by year published

Figure 3.2 shows two side by side Bar charts. The left chart shows the number of total classified fake news articles in the dataset per year, while the right chart shows the total classified true news articles per year. The total fake news seems to be much higher than the true news entries almost every year. The total is abnormally high, comparing to the rest, in the year 2017 with almost 190K results for fake news articles. This phenomenon is consistent with the results of Figure 1.1, were in the year 2017, we saw an extremely high amount of requests for fake news on the Google search engine. According to that, the web crawlers found a higher amount of articles on the web to extract. The spread of misinformation seems to have continued through 2019, where fake news articles take 88% of entries on that year. The difference is seen more clearly on Figure 3.3. Here, both fake and true entries in the dataset are shown in the same chart to better visualise their relationship.

The set of Scatter plots, shown across Appendix A.1, visualise the total of fake news articles produced by the different publishing domains in our dataset for each year. To better show this data, we split it into ten subplots where we divided the domains by calculating the median of their total entries through the decade to have more close results in each subplot. The median, also called "the middle" value, is the value separating the higher half from the lower half of a data sample. To find the median, we list the values in ascending numerical order, from smallest to largest and extract the list's median value. Each dot in the graph represents the total entries of a domain for that specific year. The lines connecting those dots show how that domain developed over the years.

We can see several domains with a high amount of fake news articles that kept misinforming their readers throughout the decade, such as redstate.com, chron.com and hollywood.com. These sites show an obscure amount of fake news activity and possible intentional disinformation of their readers. There is also a higher number of domains in 2017, as shown from the previous figures.



	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
False	12014	246	16701	10862	1414	46593	2743	17774	189168	5146	69247
True	39	22	22199	1487	816	4683	327	4275	166704	51	9366

Figure 3.3: Fake and True articles classified by year published (side by side)

Chapter 4

Implementation: Fake News Identifier

Contents

4.1 Methodology Overview

As previously discussed, part of this research is creating a system that can classify a given article as true or fake news by cross-checking it with articles from fact-checking domains with a similar claim. Our methodology for creating such a system is based on five major components: Article Data Extraction, Preprocessing of Article Content, Article Claim Extraction, Web Scraping and Near-Duplicate Detection. To better visualise our structure, Figure 4.1 shows an overview of the architectural design used the system's development process.

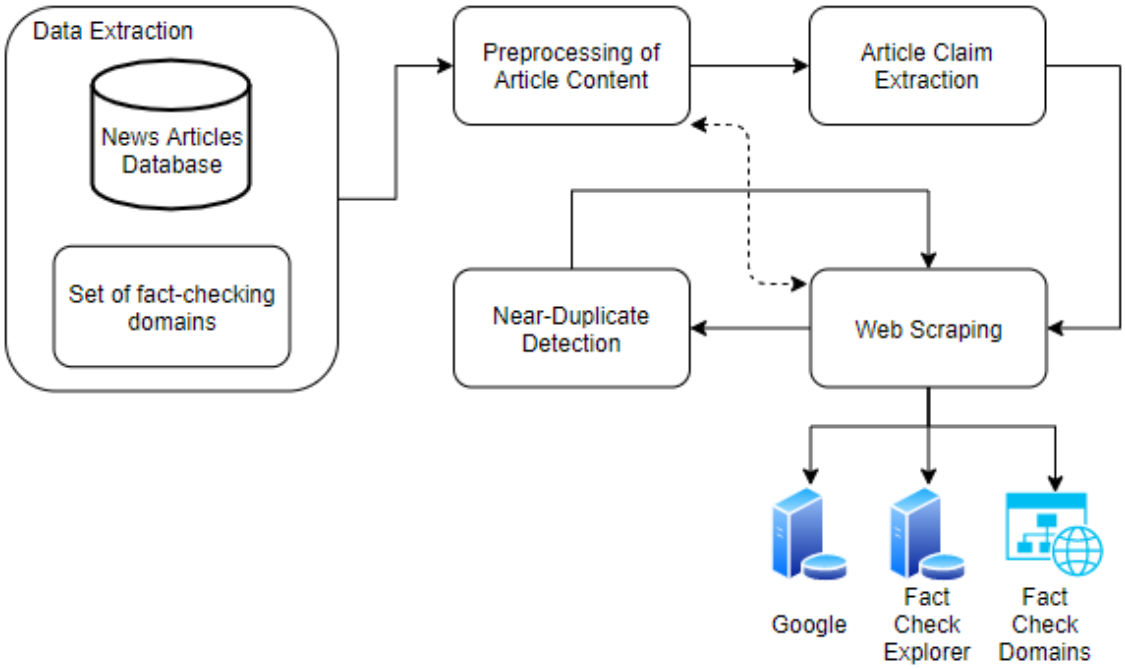


Figure 4.1: An overview of our methodology

This work aims to contribute to the growing and ongoing need to detect and reduce fake news over the Internet. More specifically, the implementation of this programming work aims to create a fully functional, applicable and integrated system that cross-checks fact-checking domains for given articles and use our results to produce an evaluation of the growth and the evolution of fake news on the Internet. To achieve this goal, it is necessary to utilise auxiliary algorithms, the selection of which is based upon previous similar studies and the evaluation of their efficiency experimentally with comparative input data used in the implementation of this work.

Firstly, articles are given by a dataset (more about the dataset used in Section 5.1), or a user, along with a set of known fact-checking domains to use as a guideline for our search. We have manually integrated the following as our system's default domains: Politifact[1], Snopes[2], FactCheck[3], FullFact[4] and MediaBiasFactCheck[5]. We will refer to the article the system is processing as the "original article". A processing of the original ar-

[1] https://www.politifact.com/
[2] https://www.snopes.com/
[3] https://www.factcheck.org/
[4] https://fullfact.org/
[5] https://mediabiasfactcheck.com/

ticle's content is then conducted to remove all unwanted data and prepare it for the next phases. After that, we extract the given original article's claim to use as a query in the Web Scraping phase so we can isolate the search spectrum as much as we can. We then search the web using the produced claim as our query. First, we search Google plainly, while isolating the results for the fact-checking domains given at the beginning. If no results are found, we search the Google Fact Check Explorer[6] [7], a tool developed by Google that searches various fact-checking domains for news articles. Finally, if no results are found, we search each of the given domains separately using their integrated search engine. For each result found we need to determine if it is referring to our original article or a near-duplicate one. The sheer number of available articles and duplicates containing fake news makes it virtually impossible to compare their entire content to verify similarity, as the process would be too time-consuming. For this reason, effective algorithms that create fingerprints of the said documents needed to be utilised, which will examine their similarity fast and efficiently. For our purposes, the Simhash algorithm was selected. A bit-to-bit comparison of the fingerprints using the Hamming Distance measurement can validate the two documents comparison. We go on to explain each of these steps in more detail.

The system's implementation process is characterised by some essential milestones that summarise the system overview shown by Figure 4.1. We go on to explain each of these steps in more detail.

4.1.1 Article Data Extraction

Given the original article, the implementation process's first challenge was to understand the data format and modify them for their proper use in extracting data from the articles. The article's URL address was used to extract only the article's necessary information (such as the title and its content) for each request. The extraction of this data was achieved using the Newspaper3k[8] library API of the Python programming language. This library is explicitly designed to extract and curate online articles. The Internet Archive[9] was also used as a backup to refer to older saved versions of the Internet if an article URL address

[6] `https://toolbox.google.com/factcheck/explorer`

[7] `https://developers.google.com/fact-check/tools/api`

[8] `https://newspaper.readthedocs.io/en/latest`

[9] `https://archive.org/about/`

is no longer available. The data is extracted and stored to be used in the following phases of the process. The above steps are used for the original article's data extraction. They will also be used on all possible duplicates we will identify at the Web Scraping phase on Section 4.1.4.

4.1.2 Preprocessing of Article Content

For the most successful processing of the data by the algorithms, it is necessary to eliminate the unnecessary "noise". For the proper preprocessing the text needs to be subjected to certain refinements explained bellow:

1. Convert all characters to lowercase.

2. Tokenisation.

   Tokenisation helps us split the text into tokens of words. For our purposes, we split them into tokens of overlapping phrases consisting of n number of words, called n-shingles.

3. Removal of white space characters.

4. Removal of Stopwords.

   Stopwords are words that often appear in texts but are unnecessary for data processing if they do not express any meaning (such as and, of, in, the, is, that, these).

5. Stemming and Lemmatization.

   These are techniques aimed at retaining the root of the meaning of words. Stemming transforms words into their original form (e.g. tokenisation $\rightarrow$ tokenise). Lemmatisation groups together the inflected forms of a word to analyse them as a single term called the word's lemma (e.g. am, are, is $\rightarrow$ be).

The NLTK library[10] of the Python programming language was used to implement the stopwords, Porter Stemming and Lemmatization techniques to avoid repetitive calculation of similar information.

[10] `https://www.nltk.org/`

4.1.3 Article Claim Extraction

The next step is to extract the core claim of the original article. Two approaches were implemented and tested. The first approach uses the requested article's title after a noise elimination preprocessing to remove any unnecessary punctuation characters, white spaces or repetitions. The second approach uses the TextRank algorithm [20], often used in keyword extraction and text summarisation. We use this algorithm to extract the most important sentences of the text article.

The TextRank is similar, and based, on the PageRank algorithm [10], which is used to calculate the weight for web pages. The PageRank is visualised as a directional graph of webpages and their links to other web pages and they can be assigned a weight using the following formula:

$$S(V_i) = (1-d) + d * \sum_{j \varepsilon In(v_i)} \frac{1}{\left|Out(V_j)\right|} S(V_j)$$

- $S(V_i)$ - the weight of webpage i
- d - damping factor, if there are no outgoing links
- $In(V_i)$ - set of inbound links of i
- $Out(V_i)$ - set of outgoing links of i
- $\left|Out(V_j)\right|$ - number of outbound links

As shown by the equation, the weight of a webpage is dependent on the weights of its inbound web pages. This iteration is run miltiple times to get the final weight.

For the TextRank algorithm to find the most relevant sentences, the web page is represented by the text and follows the same ideology as in the PageRank. We split the document into several sentences and tokenise their words to generate a list of tokenised sentences. We then build a Similarity matrix between the sentences, calculated using the Cosine Similarity metric, as shown in the equation below, to measure distances between documents irrespective of their size.

$$similarity(A,B) = \frac{A * B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

We can now run the PageRank algorithm on the similarity matrix we have produced and generate a PageRank matrix that will hold the score for all the sentences, and the sentence with the highest score is considered the most important.

We use the highest-ranking sentence to extract its core with the use of semantic triples. A triple is a set of three entities that codifies a statement about semantic data in the form of subject-predicate-object expressions, as shown in Figure 4.2. The Stanford Core NLP library was used for the implementation [9]. The system splits the sentence into a set of entailed clauses. Each clause is then shortened even more and finally produces a set of entailed shorter sentence fragments, semantic triples.

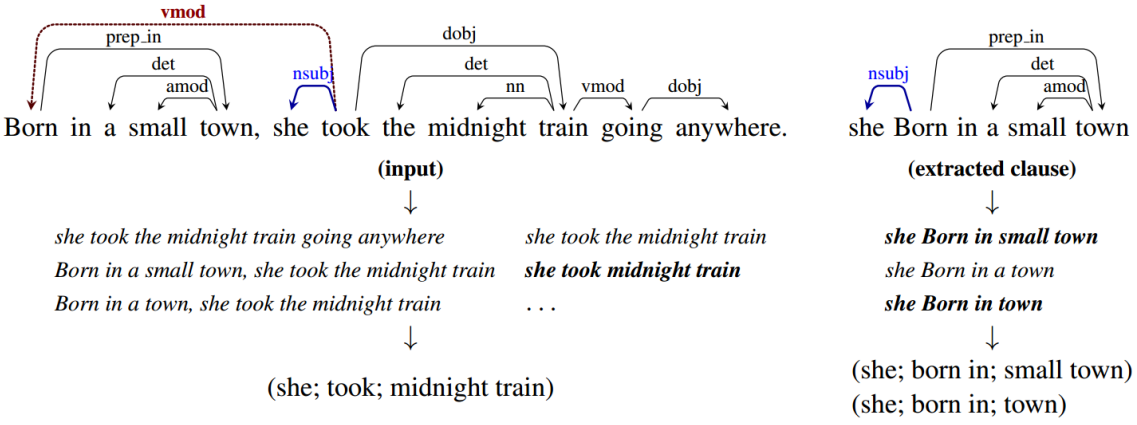


Figure 4.2: Semantic Triples: Overview

A small sample of articles was preselected and manually evaluated to test these two methodologies, and how effected they were when used as a query for the web scrappers. The second approach with the TextRank seemed to have performed slightly better than the first but has the drawback of being more resource and time demanding, due to the need of more calculations and the connection it has to keep with the Stanford server to keep the pipeline active for the NLP Core library to work.

For those reasons, the first approach using the preprocessed original articles' title is used in the final implementation.

4.1.4 Web Scraping

Next, the implementation process predicts the identification of fact-checking domain articles that have evaluated the original article's authenticity and reliability so we can classify the original article as true or fake. To collect these articles across the World Wide Web, we use the claim, found in the previous step of Article Claim Extraction in Section 4.1.3, as our query. For its implementation, we first tested the Google API[11] for article recovery from the Google Search Engine. While this tool was free and registered as the official Google API for web searching, it came with restrictions. Due to its limitation of not allowing us to retrieve many items too quickly, it was condemned as not suitable for our current work.

For the final implementation, different web crawlers were implemented with the BeutifulSoup library[12] for the search engines' result collection. For each search query, the claim is used as the query. A range restriction is implemented to search only for articles published close to the original article's publication. The search is done in three different ways to widen the search spectrum to find a result:

1. Google Searching

   We are searching for articles on the World Wide Web using the Google search engine. A Web Crawler searches for the search engine's first n results and extracts the links using the BeaurifulSoup library. Each link is then evaluated if it is published under the set of fact-checking domains given to us as input on the Data Extraction phase on Section 4.1.1.

2. Google's Fact Check Explorer Searching[13] [14]

   Again, using a Web Crawler, we get the top results of Google's Fact Check Explorer. This tool's API also gives us the option to isolate the search to a specific domain from our given input list of domains and to a range of dates to match the original article's.

[11] `https://github.com/googleapis/google-api-python-client`

[12] `https://www.crummy.com/software/BeautifulSoup`

[13] `https://toolbox.google.com/factcheck/explorer`

[14] `https://developers.google.com/fact-check/tools/api`

3. Fact-Checking Domain Searching

   If all others fail, we also search each of the domains from their integrated webpage search engine. Some of them can be implemented using a simple BeutifulSoup crawler. However, others use third-party search engines where the Selenium framework was needed to collect the results, because of the extra delay required for the search engine to return its findings.

Immediately after retrieving the results' URLs, it is necessary to extract the articles' necessary data (such as their title and content). The Newspaper3k API is used once again, as shown in Section 4.1.1 for the Data Extraction process. The result is then being preprocessed using the techniques shown in Section 4.1.2 to prepare it for the next step of Near-Duplication Detection.

4.1.5 Near-Duplicate Detection

The final, and arguably the most critical step of the process is the comparison of the content of the original article with that of the candidate duplicates retrieved from the World Wide Web in Section 4.1.4. For each result, which points to a fact-checking domain's evaluation of an article, we need to determine if it refers to our original article or a near-duplicate one. We have implemented three techniques to give an approximation of that answer as best as we could.

First, we check if our fact-checking domain result is redirecting in any way to the original article by checking the hyperlinks of the article and its sources. If this is true, then the article is most probably an evaluation of the original article and therefore a correct result for our detection process.

The second technique uses state-of-the-art algorithms for near-duplication detection of text documents. Two approaches were used and tested for their detection. In the first approach, the Minhash algorithm was utilised to create signatures for each article and the LSH (Locality Sensitive Hashing) algorithm for identifying similarities between the articles with the calculation of distances to be done using the Jaccard Similarity. In the second approach, the Simhash algorithm was utilised for creating signatures for each ar-

ticle and the Hamming Distance to calculate their distances. A detailed description of how the algorithms were implemented and how they work follows, along with the way we have evaluated and compared them.

Finally, suppose the two previous approaches fail. In that case, we use the TextRank algorithm to extract the top keywords from the articles and determine if a large percentage of those keywords are included with similar weight in both articles. This method is experimental. Due to time limitations, it was not fully optimised but still held a decent precision score in the later evaluation of the system (Chapter 5).

Following is a detailed description of how our algorithms were implemented and what parameters we chose for our experiments. On Section 4.1.6, we describe how we have evaluated and compared our two approaches for the similarity algorithms.

Minhash Algorithm

For the implementation of the Minhash and the LSH algorithms [2] the Snapy[15] Python library was utilised. Initially, large-scale data management creates multiple issues, such as the inability to fit the entire text to be examined in the main memory. The approach implemented to deal with this issue requires converting the text in smaller sets, dividing it using the logic of n-shingles (or n-gram). More specifically each shingle is a sequence of n tokens which appear in the article. In this case, the tokens are words and n is an empirically preselected constant number for the algorithms' execution.

The Minhash algorithm is utilised, whose primary purpose is to convert large sets of text into short text signatures. Therefore, the shingles produced contain the articles' entire content and are utilised by the Minhash algorithm resulting in small-scale vectors representing the original text while maintaining the necessary similarity.

The Minhash algorithm is based on the presence of one boolean input panel of Shingles and the to be examined articles. The presence of a shingle in an article is denoted

[15] `https://pypi.org/project/snapy/`

by 1. Each article mutates through a random permutation π, and this is achieved with the utilisation of a "hash" function that expresses the following methodology: "The number of the first line in which column C has the value 1", as shown in the following equation $h_\pi(C) = min_\pi(C)$. The value C represents one feature of the text article.

Figure 4.3 represents an example of the Minhash algorithms execution and its resulted signature. After executing Minhash on each candidate article and producing their signatures, we are left with a set of candidate signatures and the original article's signature whose duplicate we are trying to find. To verify their similarity, we run the LSH algorithm, which works with the philosophy that focuses on pairs of signatures that are likely to belong to similar documents. According to LSH, the columns of matrix M, shown in Figure 4.3, containing the candidate signatures are split into many buckets so that the articles that are in the same bucket represent possible duplicates or candidate pairs.

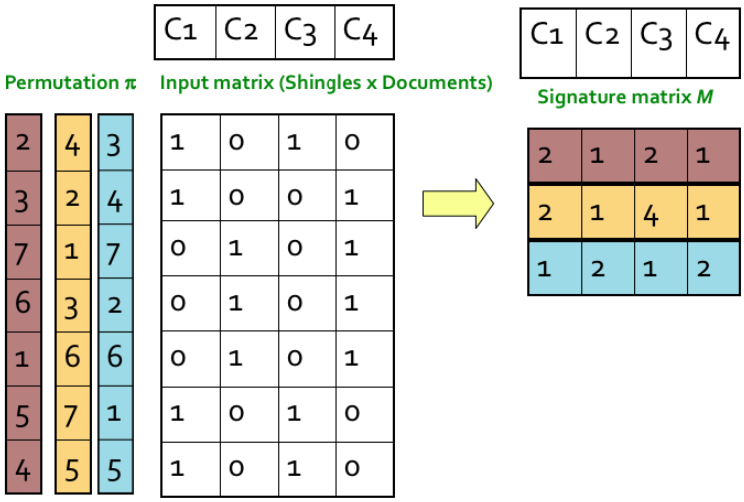


Figure 4.3: Minhash Algorithm: Overview

To find those candidate pairs, given a matrix M from the Minhash algorithm, we select a value s (fraction < 1) indicating the similarity threshold. More specifically, the x and y columns of the matrix M represent a candidate pair if and only if their signatures agree for at least s percentage of their rows. The relation is given by the equation $M(i,x) = M(i,y) for at least frac. s values of i.$

The LSH algorithm divides the table M into smaller parts called bands, to achieve displaying only the similar columns in a bucket with a high probability of similarity. The columns of matrix M are hashed multiple times. Each band consists of a fixed number of lines. Additionally, for calculating the signatures' similarity distances, the Jaccard Similarity metric was utilised because it is considered ideal for the Minhash and LSH algorithms.

The variables used for our implementation were selected from suggestions of other researches and experimental tests. A word tokenisation for 3-shingles was conducted using prime numbers to create the hash function for 100 permutations under a number of 50 bands to break the minhash signature into buckets with no set threshold.

Simhash Algorithm

For the Simhash algorithm's implementation, the Simhash Python library[16] was utilised, which is based on the algorithm proposed by Charikar in his research [3]. This method is based on dimensionality reduction, converting a large-scale matrix into small f-fingerprints.

After a preprocessing of the document by n-shingle tokenisation, just like with Minhash, the algorithm follows the following procedure to create that f-fingerprint, also shown in Figure 4.4.

An f-dimensional matrix, called V, is created where each of its values is first set to 0. Each value is hashed by an f-bit hash-function, creating an f-bit unique feature. These features make the matrix V as follows: if the i-th bit of the hash function is 0, then the i-th value of matrix V will increase its weight by one, and if the i-th bit is 1, then the i-th value of matrix V will decrease its value by one. When all features are processed like above, matrix V will consist of positive and negative values. Each negative value will be modified to 0 and each positive value to 1. We are then left with an f-fingerprint which describes the original article.

[16] `https://pypi.org/project/simhash/`

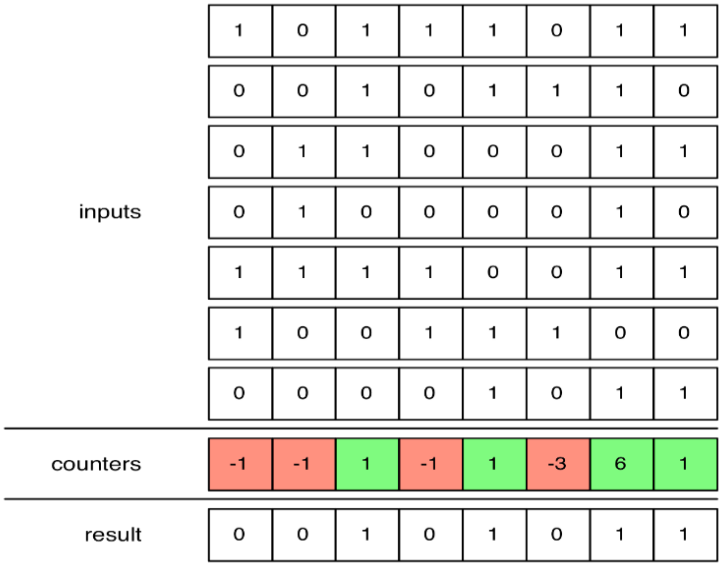


Figure 4.4: Simhashing Algorithm: Overview

After running the Simhash algorithm and collected a set of f-fingerprint for each candidate article and an f-fingerprint for the original article F, we want to detect if any of the fingerprints in the set differ from F at most k bits, with k as our similarity threshold. The Hamming Distance is utilised, a technique for measuring the distance from two binary tables, where it measures how many bits they differ. Two articles, A and B, are near-duplicates if their Hamming Distance d(A, B) is less than or equal to k, $d(A,B) \leq k$.

The variables used for our implementation were selected from suggestions of other researches [3, 8] and experimental tests. A word tokenisation for 3-shingles was conducted for the creation of 64-bit Rabin fingerprints. The threshold of 13 was selected after an evaluation analysis was conducted. Figure 4.5 was created after a human evaluation of a dataset of near-duplicate articles, more on that on the Algorithm Comparison of Section 4.1.6, visualising our experiments precision and recall. The precision is defined as the fraction of our model's True Positive results, meaning the pairs whose Hamming distance was at most k. Our True Positives (TP) are the number of accurate predictions made by our algorithm, and False Positives (FP) are the number of inaccurate predictions found falsy as true near-duplicates.

$$Precision = \frac{TP}{TP+FP}$$

The recall measures how well we predicted the positive results when they were actually positive. False negatives (FN) describe the inaccurate predictions that were genuinely negative.

$$Recall = \frac{TP}{TP+FN}$$

Figure 4.5 shows the compensations for various values of k, where a too-low value shows many False Negative results, and a too-high value indicates a lot of False Positive results. Our choice for k is $k = 13$ based on the above where recall and precision intertwine close to each other. To summarise, our sample for 64-bit fingerprints declares two documents as near-duplicate when their fingerprints are at most 13 bits different.

The relatively high value of k can be attributed in the presence of noise that was not removed during the preprocessing of the data before executing the Simhash algorithm. The noise in which we are referring to may include: links referring to other websites, text from cookies or privacy policies incorrectly extracted by the Newspaper3k API instead of the articles' content and more. In addition, in some cases the large difference in the length of the original article's contents with the possible candidates may leed to inefficient operation of the algorithm, producing a false result.

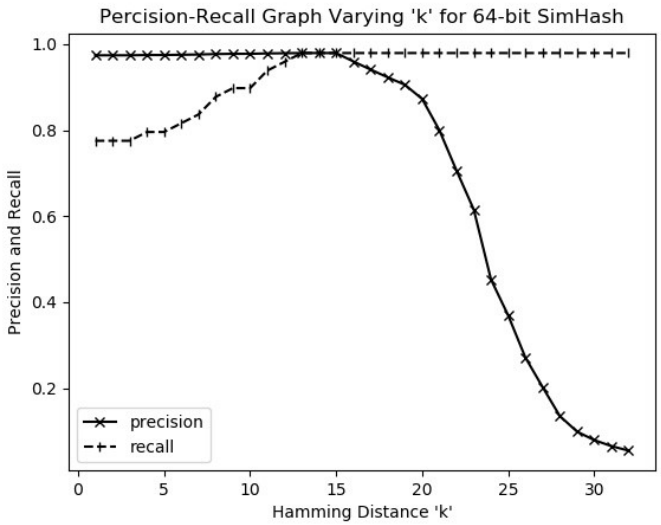


Figure 4.5: Simhashing Algorithm: Precision-Recall Graph for different k

Keyword Extraction

The final technique uses the TextRank algorithm [20], described in the claim extraction on Section 4.1.3, to extract the articles' most important words. Instead of the web page, we have in PageRank [10], TextRank is represented by text but follows the same ideology. We split the document into several sentences and, since not all words are useful to us, we store words with the specific POS (Part-of-Speech) tags of NOUN, PROPN and VERB, using Python's Natural Language Toolkit[17] (NLTK). Each word is the same as a node in the PageRank algorithm, so our nodes are represented by the words $w_1, w_2, w_3, ..., w_n$. By setting a window size of k, $[w_1, w_2, ..., w_k], [w_2, w_3, ..., w_{k+1}], [w_3, w_4, ..., w_{k+2}]$ represents our windows and two-word pairs in a window are an undirected edge on our graph. By following the PageRank algorithm's equation and running it multiple times, we can calculate each node's weight (word) and find the important words, called the keywords.

After extracting the keywords, we pick the top n keywords and calculate if at least k of these keywords appear in both articles with a precision factor of +/- m. In short terms, we are trying to find if a large percentage of those keywords are included with similar weight in both articles and therefore describe a similar claim. In our system, we used the 10 top keywords, where at least half of them should appear in both articles and may have a weight difference of 60% or less. This feature was not fully developed and optimised due to time limitations, and the variables were selected from experimental tests.

4.1.6 Evaluation of Similarity Algorithms

Similarly to the research Finding Near-Duplicate Web Pages: A Large-Scale Evaluation of Algorithms [5], we compared the similarity algorithms of Minhash (Section 4.1.5) and Simhash (Section 4.1.5) to picked the best one to utilise in our system.

The FakeNewsNet open-source dataset [14–17] was utilised for the evaluation and the comparison of the two algorithms. This dataset consists of preselected true and fake articles from the PolitiFact fact-checking domain, containing political news articles, and the Gossipcop domain, containing entertainment and gossip articles. The dataset's purpose

[17] `https://www.nltk.org/`

is "collecting, analysing and visualising fake news and related dissemination on social media". The dataset contains multiple CSV files that follow the structure, as shown in Table 4.1.

Columns	Description
_id	Unique identifider for each news
url	Url of the article from web that published that news
Title	Title of the news article
tweet_ids	Tweet ids of tweets sharing the news. This field is list of tweet ids separated by tab.

Table 4.1: Columns of the new FakeNewsNet dataset

The *tweet_ids* tag was not used because we are focusing on web news articles and not social media feed from Twitter. For the data extraction the Newspaper3k[18] library was used once again in cooperation with the Internet Archive[19] as we have shown in Section 4.1.1. The preprocessing of the articles' contents also follows the same process as in Section 4.1.2. For the claim, the *title* tag from the dataset is being utilised. As for web scraping the web for near-duplicate articles, only the *Google Searching* technique was used from Section 4.1.4.

Possible near-duplicate articles were identified for each entry of the dataset using the Google search engine's first page for querying. A human evaluation was conducted from a sample of results where each pair of possible near-duplicate articles was classified as:

- Near-Duplicates

  The pairs whose signature was under or equalled the algorithm's threshold.

- Non-Near-Duplicates

  The pairs whose signature was over the algorithm's threshold

- Undefined

  These articles might have had a false content extracted from the Newspaper3k API or linked to a video, image, privacy policy or cookie warning by accident.

[18] `https://newspaper.readthedocs.io/en/latest`

[19] `https://archive.org/about/`

Minhash & LSH Approach			
Datasets	Duplicate Rate	Median	Max
Politifact	1.322	1	4
Gossipcop	1.256	1	11
Both	1.260	1	11

Table 4.2: Minhash and LSH Algorithm Statistics

Simhash Approach			
Datasets	Duplicate Rate	Median	Max
Politifact	2.259	2	8
Gossipcop	2.077	1	18
Both	2.092	1	18

Table 4.3: Charikar's Simhash Algorithm Statistics

We calculated the duplication rate for the two approaches, showing the mean number of near-duplicate articles identified for each article in the dataset, since one article may have multiple near-duplicates.

For each of the two algorithmic approaches, a separate output file was generated with results. Each URL address of the original articles is linked to a different URL address of a possible near-duplicate article, verified according to the algorithmic approach's comparison method. These collections were used for the evaluation and comparison of the algorithms. The Tables 4.2 and 4.3 show the results for the data between the two approaches. We also show the difference between the Politifact and the Gossipcop published articles. The median and max total near-duplicates are shown for each of the above to better understand our results.

It is clear that the Simhash method had higher success in locating near-duplicate articles in both political and gossip articles. It is worth mentioning that the maximum number of near-duplicate for an article was by far higher for the Gossipcop dataset, indicating that a news item with false gossip content was reproduced an extensive amount of times. Regarding the duplication rates and median numbers, they are relatively close to one another,

showing that we get a similar number of duplicates for an article regardless of content. Lastly, the tables also show that the total amount of political news articles had a higher rate of near-duplicate articles, possibly because of their higher interest and demand from readers than gossip articles.

With the above findings, we have chosen Simhash as the algorithm to be used in our system for near-duplicate detection. The article's content is tokenised into 3-shingles with a threshold of 25. The threshold had to be increased because, in contrast with the articles used to compare the algorithms, the articles our system is comparing are from fact-checking domains where they evaluate the original article by providing sources and arguments. So it is expected that the two articles may need a slightly higher similarity threshold.

4.2 Annotation System

After our system was fully developed an annotation website was needed to host our system for execution and as a crowdsourcing tool to fully evaluate our system. This website needs to follow some usability heuristics for the best user experience while using the system. It needs to be easy to use, provide feedback to the user when performing a task and be secure and respectful to the university guidelines. The final application was host using the Surge[20] static web publishing tool and can be accessed when connected under the University of Cyprus domain with a direct or VPN connection, using this link `http://fake-news-dissertation.surge.sh/`.

4.2.1 Front End

Our web application was created using the open-source NodeJS[21] cross-platform runtime environment that executes JavaScript code outside of the web browser, designed to build scalable network applications. For the implementation of our web application's user interface, the ReactJS[22] library for JavaScript was utilised for the creation of a Single-Page-Application (SPA). The React library offers us the ability to represent parts of our code

[20] `https://surge.sh/`

[21] `https://nodejs.org/en/`

[22] `https://reactjs.org/`

as components integrated based on a specific hierarchy between them, thus resulting in a clear and user-friendly structure in our application. Each component consists of exclusive features and states responsible for renewing and initiating their component only when rendered on the screen. Components are written using JSX, a syntax extension to JavaScript designed to understand and process HTML code inside the JavaScript language.

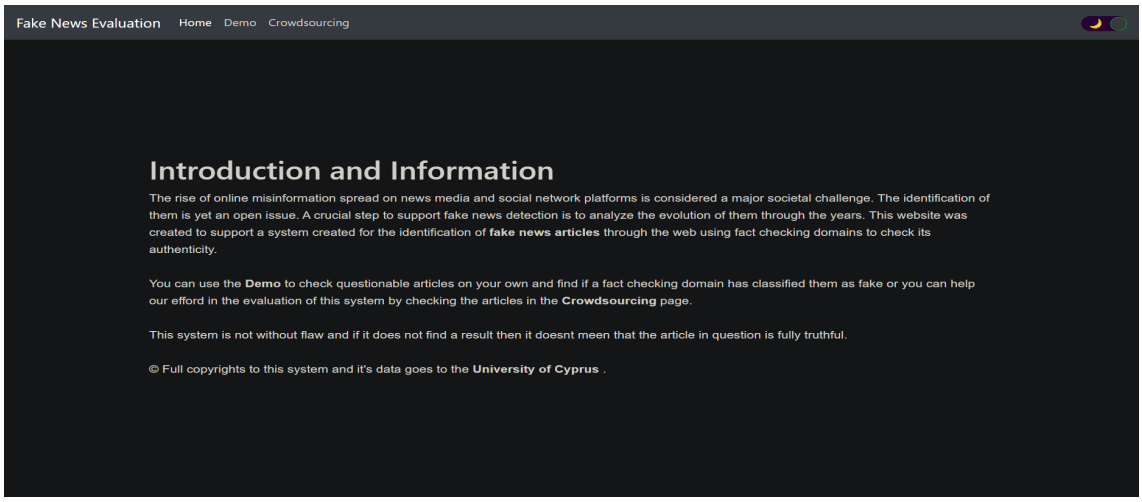


Figure 4.6: Annotation System: Home

Three pages were implemented to satisfy our needs. The Home page, shown in Figure 4.6, is the first page you see when entering the application. It displays a short description of this application's capabilities and why it was created. The users can use the navigation bar on the top of the page to move between our other pages and change the application theme from dark to light using the slide button on the top-right side (Figure 4.7).

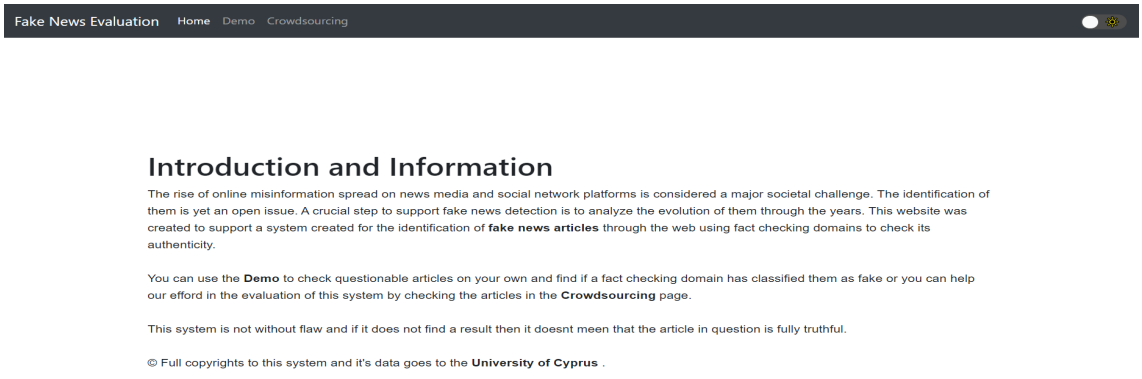


Figure 4.7: Annotation System: Home - Light Theme

When navigated to the Demo page (Figure 4.8), we can see the interface designed for the user's communication with the system. Here they can run URL addresses of suspicious articles with the hope of identifying them through a fact-checking domain. The user sees a form that they can fill to feed the system the data it needs to execute their request.

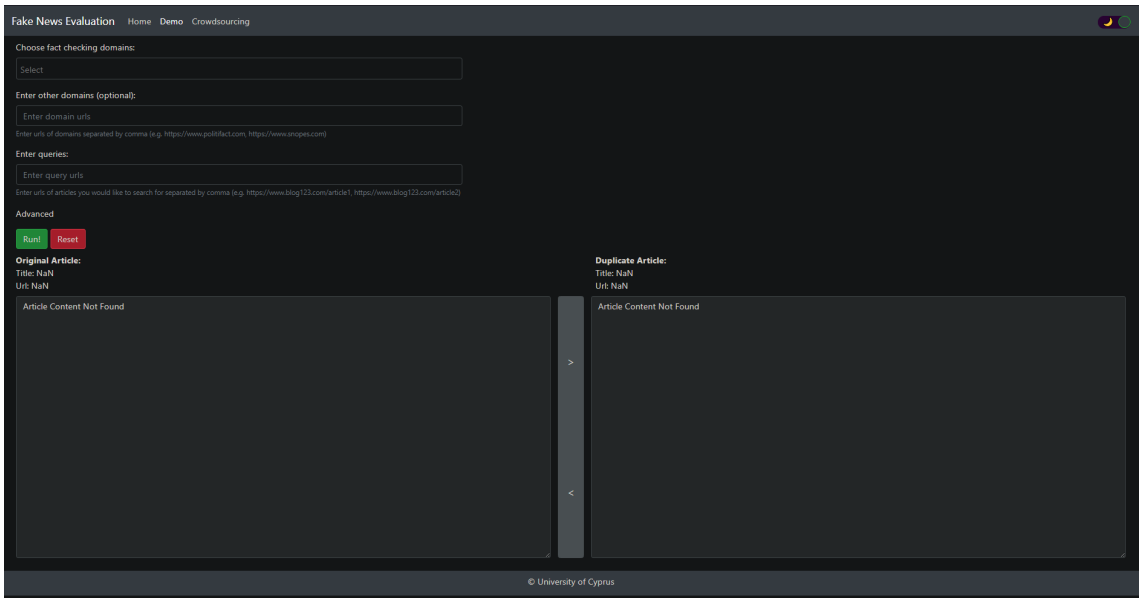


Figure 4.8: Annotation System: Demo

The first input is a drop-down list of fact-checking domains that we have already implemented in the system, and the user can use as many as they want to be used for the web scraping of the results. If none are selected, then the system will use all of them as per default. The second input box is optional, and the user can enter more fact-checking domains if they wish to broaden the search spectrum through the web. If more than one URL is entered, they have to be comma-separated as instructed under the input box. The last input will hold the URL of the article or articles requested for identification from the system. Like before, if more than one input is entered, they have to be comma-separated. If the user is more experienced with the use of similar systems, they can use the Advanced feature to change the system parameters to whatever they prefer. The default values are the same ones we used when running the system to gather our results. After the user is done filling the form, they can run the system by pressing the RUN! Button or reset the form and start again. An image of this process can be seen in Figure 4.9.

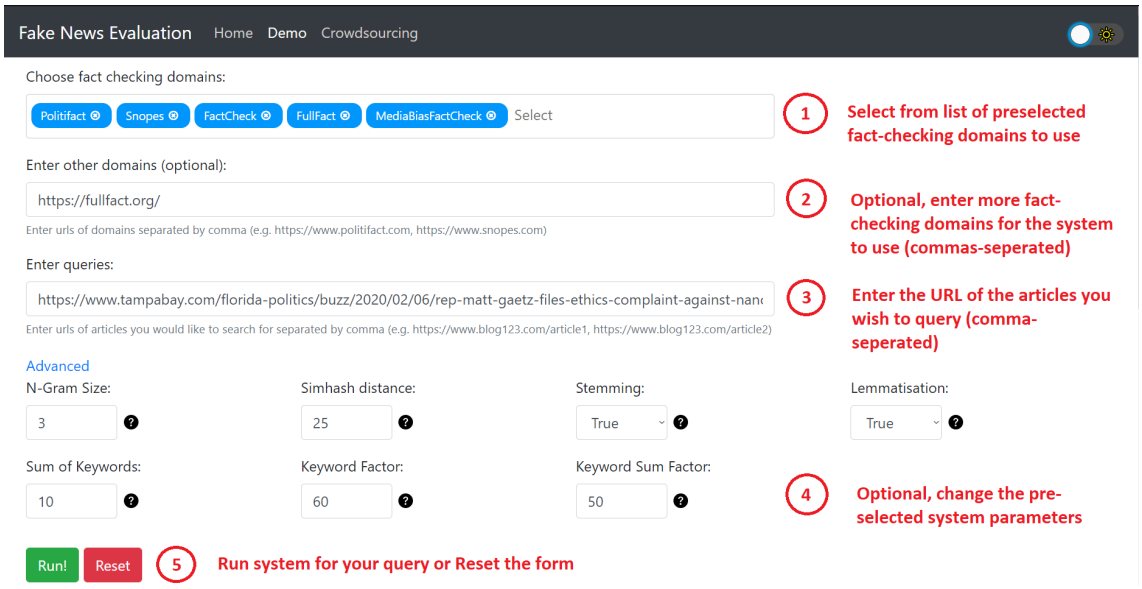


Figure 4.9: Annotation System: Demo - Instructions

We have implemented some helpful features for the better user experience to help the user know how the Demo works and what state it is currently executing. Next to each parameter's input box, in the Advance section, we have created a helpful note for the user with a short description of each parameter, as shown in Figure 4.10. When the system begins executing their request, the system's state is shown in the run button's place to show how many of the requested articles are done executing, so the user knows that the system is still processing and has not run into an error. Finally, appropriate error messages are shown in the form of a browser warning if the system runs into any problems, like if there is no connection to the system's API.

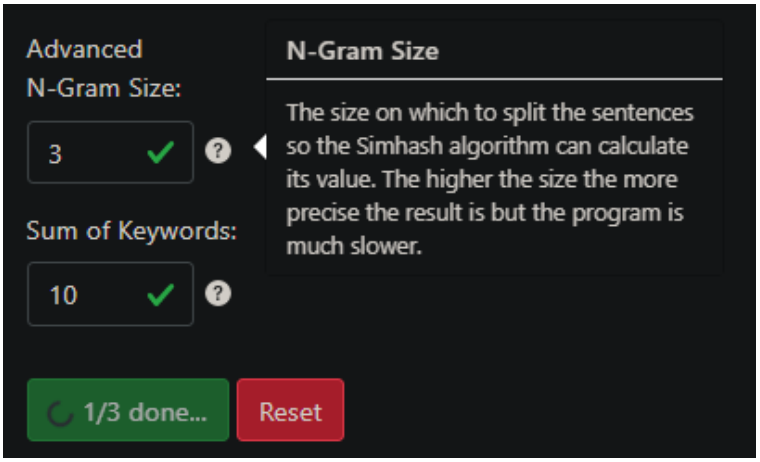


Figure 4.10: Annotation System: Demo - Help

When the system is executed, the results are displayed in the big text boxes at the bottom of the screen, as seen in Figure 4.11. The left box represents the original article, as requested by the user, and the right one is the result found by the system. If no result is found, an appropriate text is written instead. Along with each of the articles content, their Title and URL address are also displayed. When the user has requested multiple articles for identification, they can use the two arrow buttons, between the text boxes, to navigate them.

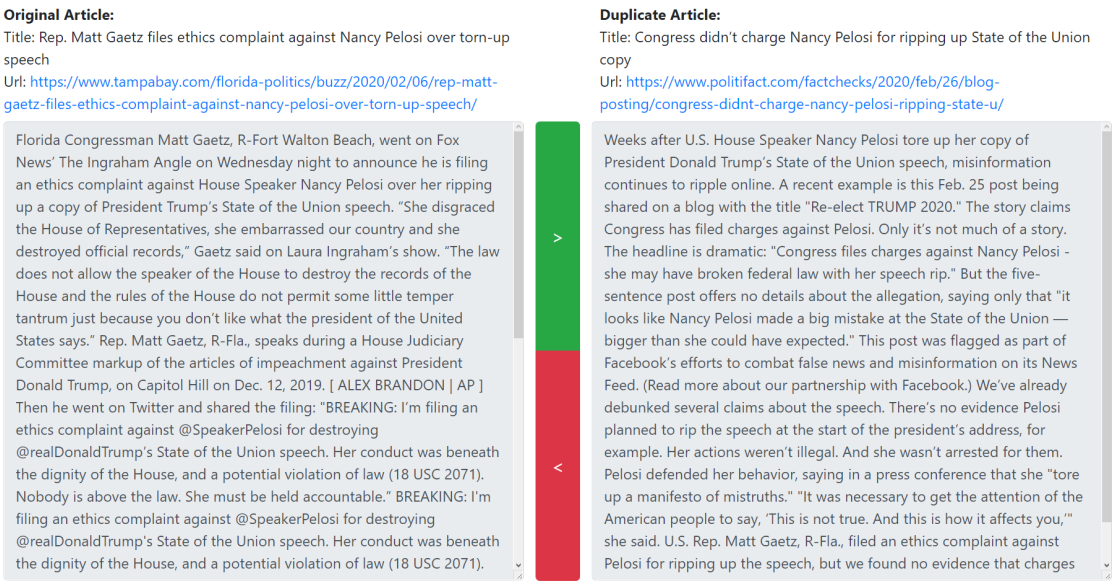


Figure 4.11: Annotation System: Demo - Results

The third and final page is the Crowdsourcing page. Our crowdsourcing team uses this page to evaluate our results. More on that on Section 4.2.3.

4.2.2 Back End

The website, which was implemented as a NodeJS[23] project with the ReactJS[24] library's use, had to communicate with the system written in Python and collect the result in Javascript to display the information to the user correctly. The system needed to remain under the University of Cyprus domain to communicate with the crowdsourcing evaluation process database and stay secure under the university's firewalls.

[23] `https://nodejs.org/en/`

[24] `https://reactjs.org/`

An API was created using the Flask API[25], which is also written in Python and can run the system in the background. The annotation website communicates with the API using restricted and secure HTTP requests. While this makes the system slower, it keeps it protected because it needs to communicate with the API. After the API receives the HTTP request with the chosen fact-checking domains, the requested query URL, and the system parameters' values, the fake news identification system begins its execution. After all the results have been gathered, they are returned to the user, again using HTTP requests. The web application receives the data and displays them to the user as we saw in Figure 4.11.

Because the API runs under the university's domain, to use it, users need to be connected to the UCY domain, either directly or with VPN to run it.

4.2.3 Crowdsourcing

Our resulting dataset used in our research evaluation process was assessed using a combination of manual and crowdsourcing annotation efforts. We will later use this dataset to conduct several exploratory analyses to identify our system's performance in detecting articles from fact-checking domains and extract essential conclusions about our work and ways it can improve. More about the dataset in Chapter 5.

Because of the enormous size of the dataset collected of 16.5K entries of possible identified articles, crowdsourcing was a deep necessity to finish the manual evaluation of our collection in time. Several University of Cyprus students were asked and agreed to assist on our efforts as free-agents. We explained how the manual evaluation of our data works and what they had to be aware of while deciding if the article the system identified is a proper evaluation of the original article's reliability. The fact that none of our evaluation team is a journalism major possesses the challenge of fully understanding the journalistic style of writing in articles versus the informal style they are used to read daily. Journal articles are usually lengthier than consumer reviews and opinions, thus increasing the task's difficulty as they are required to read full news articles and compare them by identifying both their core claim. Since there was not enough time or budget to hire news experts,

[25] `https://flask.palletsprojects.com/en/1.1.x/`

our team of students was kind enough to assist us.

As part of our annotation system, we set up a page to hold the evaluation process for our crowdsourcing team and us to easily classify our results, as seen in Figure 4.12.

Figure 4.12: Annotation System: Crowdsource

The logic behind how we display the data is the same as the results of the Demo page. There are two main boxes. The left one represents the original query that the system was asked to identify, and the right one holds the result. The articles title and URL addresses are also displayed above them. The web application asks our API to fetch the next undefined entry in our database that the system has found a result. The person evaluating the two articles has to read both of them and manually extract their claims. Suppose they deem that the system's resulted article (right) is a fact-check evaluation of the original article (left). In that case, they press the "Yes" button to update the dataset and identify it as a correct result. If not, they press the "No" button to identify it as a false-positive result. After a pair has been evaluated, the next one on the list is brought forth. The evaluation process took a whole month and hundreds of working hours to complete.

Chapter 5

Evaluation

Contents

5.1 Data Collection

In order to test and evaluate our system, a dataset with a plethora of fake news articles was needed. This dataset should have an already pre-evaluated collection of articles that we can use to test our system's detection rate and later evaluate our results with a crowdsourcing team for human evaluation, as discussed in Section 4.2.3. We continue to explain the data used in the evaluation phase, how we prepared it, and how we visualise our finding.

5.1.1 Data Overview

The dataset used to gather our results is the optimised collection we have extracted from the open-source "Fake News Corpus"[1] dataset, which we have already described in-depth in Chapter 3. This dataset consists of thousands of news articles gathered from trusted

[1] `https://github.com/several27/FakeNewsCorpus`

and untrusted sources published from 2009 to 2019. The "Fake News Corpus" dataset's structure is shown on Table 3.1. For our purposes, a random sample of 30K entries of fake news articles was collected. It was not feasible to run the system for all the dataset entries due to time and resource limitations.

Our system's results are stored as part of a new collection consisting of a pair of two articles. The original article used in the identification process and the possible duplicate found and returned from a fact-checking domain article. The *isDup* variable can have a value of Undefined, True or False. The above helps the evaluation system of our annotation web application recognise which of the entries are yet to be evaluated if they have a value of Undefined and the *duplicate* object is not empty, meaning that the system found a possible result for the original article. When someone evaluates a pair of articles, this variable changes to either True or False, depending on their choice of 'Yes' or 'No' buttons shown in Figure 4.12. The *similarity* variable is a number from {0, 1} describing the similarity rate found by the Simhash algorithm. Finally, the *type* variable describes which of the three near-duplicate techniques (Section 4.1.5) provided a possible result for this article and can have a value of simhash, link or keywords. The structure of the new dataset is shown in Table 5.1.

Columns	Description
_id	Unique identifier for each new entry
original	An object describing the original article, with the same features as in Table 3.1
duplicate	An object describing our systems' resulted article, with the same features as in Table 3.1
isDup	A classification of the current article pair
similarity	The similarity percentage found from the Simhash algorithm
type	The type of near-duplication technique that identified the pair as a possible result

Table 5.1: Columns of our evaluation dataset

5.1.2 Data Extraction

Our data was collected from the MongoDB dataset that stores the above collection. As mentioned before, only a sample of 30K entries was selected. The identification system was prepared with our default parameters discussed across Chapter 4. Table 5.2 also

shows a complete collection of these parameters. For each entry of the dataset our system executed, a new entry was created to the new dataset to hold the result. The system ran for two weeks, and from the 30 thousand original entries, 16.5 thousand results with duplicate fact-checked articles were found.

Parameter	Value	Description
width	3	This describes the width of the n-shingles that tokenised our text
k	25	The threshold used for our Simhash algorithm
stem	True	Porter's Stemming was used to process our text
lem	True	Lemmatisation was used to process our text
keyword_sum	10	The top 10 keywords of the text were extracted
keyword_sum_factor	50	At least 50% of the keywords should appear in both texts
factor	60	Precision factor for keywords is 60%

Table 5.2: Evaluation: Complete list of system parameters used

5.2 Results

This section presents our study results and provides a summary with the precision for each technique used to identify the articles. Moreover, we provide a visual representation of our findings, explaining the reasons behind the false-positive results, and suggesting future optimisations and a suggested threshold for a better precision score.

5.2.1 Metrics

There are multiple measurement techniques for evaluating our system's performance, such as precision, recall, accuracy and F1-score. For our purposes, we want to test the precision of our system of accurate predictions. Since we do not have an accurate classifier of our results that remain undefined, because our system did not find a candidate article to pair them with, any other metric would provide only an estimation that relies on assumptions. For those reasons, we have decided only to use the precision metric to evaluate our system's performance since it is more suited to our needs.

True Positives (TP) are the total of accurate predictions made by our algorithm that were positive, meaning all pair of articles identified by the system, and we have evaluated as True. False Positives (FP) are defined by the total number of inaccurate predictions our system identified as positive, and we have evaluated as False. Furthermore, True Negatives (TN) are described by the total number of accurate predictions identified as negative and False Negatives (FN) are all inaccurate predictions described as negative. For our model, we assume that all entries run through the system should have been paired with a fact-checked article, so every article not matched with a result is considered a False Negative.

By correctly identifying the above, we can calculate our performance metrics. The precision is defined as the fraction of our model's True Positive results divided over all relevant instances among our sample, as shown in the equation below:

$$Precision = \frac{TP}{TP+FP}$$

5.2.2 Performance

Our sample of 30 thousand articles produced 16.5K results, which is about 55% detection rate. After our manual evaluation, Figure 5.1 shows the resulted article pairs that were correctly and incorrectly identified by the system as a fact-checking domain article evaluating its original paired article, identified as the True and False Positives.

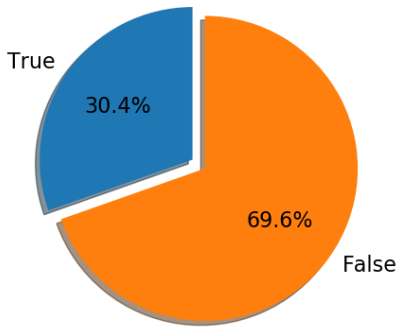


Figure 5.1: Evaluation: True and False Positive results after human evaluation

The system's performance was 30%, as derived from the results of the combination of all the detection techniques. We will discuss the difficulties the system faced and ways of optimisation. Table 5.3 provides a summary of the precision score for all three detection techniques, along with the overall precision of the system.

Detection Technique	Total results	Precision
Simhash	99.991%	0.301
Keywords	0.004%	0.672
Links	0.005%	1
All	**100%**	**0.304**

Table 5.3: Evaluation: List of the precision score for each detection technique

As we can see the Simhash method was the cause for almost all of the system's results (16.5K), with a precision of only 0.301. Although we do not have a large sample to get a more accurate estimation, the keywords technique had a precision score of 0.672. Finally, the method of detecting a link to the original article, referenced by the fact-checking domain evaluation article, had a perfect prediction rate of 1. Although the sample is too small to provide a more detailed evaluation of the technique, it was expected from the beginning that this detection method has a high precision score.

5.2.3 Optimisations

Our system did not have a high precision score, mostly because of time limitations that did not make it possible to run the system multiple times with different parameters in order to optimise it fully. The fact that two of the three detection techniques held a high precision score indicates its potential and what it can achieve with more experimentation.

First of all, we need to talk about the Simhash technique since it holds the higher percentage of the resulted article pairs found by the system and therefore is responsible for its lower precision score. In our experiments in Section 4.1.5 when explaining the Simhash algorithm, we provided the reasoning behind the parameters selected. Since the dataset used to extract the original threshold was for near-duplicate web articles and not for identifying articles through fact-checking domains, we had to increase it to compensate for

the extra information that the fact-checking domain includes in their articles. The value of 25 was selected as a threshold and 3-shingles for the tokenisation of the text.

After examining our results, we found that Simhash is weak against comparing texts with a considerable difference length, with the one being too small and the other too large. Also, if both texts are too small, then there is not enough information to make an accurate calculation. One such example was an article pair where the original article suggests ways of ordering a *pepperoni pizza*. The resulting article fact-checks a 911 call disguised as an order for a *pepperoni pizza*. We have determined that the threshold value selected can decrease to provide us with a more accurate prediction score. When the above occurs, another way of optimising our results is to use the keyword extraction method instead, which could have a higher precision score.

Several results were also falsely selected by our Newspaper3k API. Instead of the main content of the article, a warning or cookie alert appeared. Due to the Simhash algorithm's weakness for small texts, it held a falsely high similarity rate. More techniques can be utilised to prevent the detection of such messages or to identify popular keywords used in such texts as privacy-policy or cookie-warning. Furthermore, websites such as `https://mediabiasfactcheck.com/help-us-fact-check/` with a high similarity rate kept appearing throughout the results, possibly because of the search engines giving them a high priority or because there were not enough results for specific queries, since filtered for the top N results. These websites could be blacklisted to prevent them from appearing and decrease our processing time since they will not be queued for processing.

More experimentation is needed for the keyword extraction technique. The system can be executed with only that as a detection method and attempt to find a more optimised set of parameters, while also getting a more extensive data sample. The same could happen for the method of detecting the original URL address as a reference. This method should always have an almost perfect result since it was used from the fact-check domain as one of the few sources to evaluate the original article's reliability, so not much optimisation can be done.

As mentioned in the Introduction, our research will contribute to a larger scale project for Fake News Evaluation. For this system to be better equipped and make better predictions, a threshold needs to be identified to classify our results correctly. The following graph in Figure 5.2 shows the total number of FP and TN as a function of a threshold, beginning from 0.50 to 0.99 with a step of 0.1. As observed from the graph, as our threshold increases the number of true negatives increases and the number of false positives decreases. Both of them seem to change linearly, until close to 0.95, where the TN starts a more dramatic increase while the FP begins a more considerable decrease.

Our system had a 0.304 precision rate with a threshold of zero, as shown in Table 5.3. It is essential for a correct threshold to be selected to suit that project's needs for a high precision rate to be achieved. If a small threshold is selected, we will have a larger detection rate but with a low precision score that can genuinely damage our system's reliability. On the other hand, with a large threshold, a high precision score will be achieved but with a very low detection rate, making it hard for the system to produce results.

All the above suggestions should be taken into account for future optimisation of the system to improve its prediction rate and make it more suitable for research and possibly commercial use.

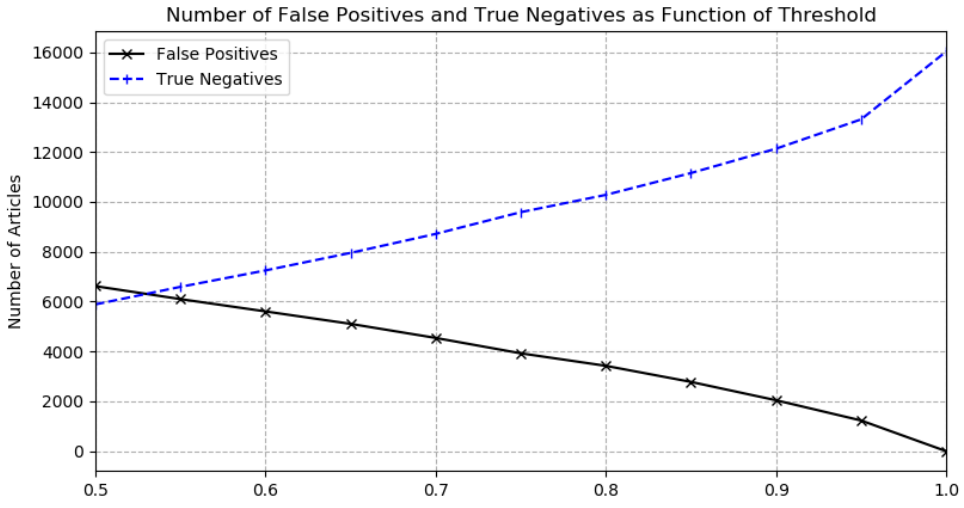


Figure 5.2: Evaluation: Number of False Positives and True Negatives as Function of Threshold

Chapter 6

Conclusion

Contents

6.1 Conclusion

This thesis aims to examine the impact of fake news on the Internet over the last decade and provide with a fake news detection system for identifying fake news articles through fact-checking domains, already expertly evaluated. We have shown that the interest in fake news has never been higher and the demand for more research and ways of fighting against misinformation is at a new peek. In doing so, we have contributed to the fake news classification challenge. We firmly believe that we have achieved our objective and gathered valuable data that will help researchers address this global pandemic of misinformation. The variety of state-of-the-art techniques used for the creation of our identification system and the datasets collected provide with ample opportunities for future work that can be done to expand the above research and significantly optimise the developed system to be used as a tool for fake news identification for both research and possibly commercial use. We are convinced that fake news detection is one of the most critical components of overcoming this challenging issue called Fake News.

6.2 Future Work

Our study approaches the challenge of fake news classification with state-of-the-art algorithms for fake news detection and natural language processing techniques. However, there is an excessive amount of more complex and sophisticated algorithmic approaches to the problem that studies have shown to be very promising and can significantly impact our methodology in improving our precision score when possibly combined with our methodology. Further future work consists of the optimisation techniques we proposed in Section 5.2.3 on how our system can be better utilised. Along with our easily programmable API, our identification system can easily be modified to suit another researcher's needs and perform more experiments for fake news.

Feature-based fake news studies aim to create systems that automatically learn from a set of collected observations and features to identify fake news correctly. By offering these features to a machine learning framework, several potentially useful patterns in the identification process can be detected. Since fake news keeps evolving in order to stay relevant and avoid further detection, the time of publishing plays a significant role in the identification. With the help of the datasets used in this research, a model could be trained to detect these patterns through different publication years using samples of data collected in that particular year. When all are combined a comprehensive set of features can be extracted to be used within a supervised learning framework that can be used to predict fake news.

Moreover, a greater analysis of fake news evolution through earlier years and years to come could provide an even more in-depth understanding. As our data suggest, fake news will continue to evolve throughout the years and adapt to defend themselves against more sophisticated detection techniques. The spread of fake news shows no form of slowing down due to humans' opportunistic and exploitive nature, which would benefit from delivering false news, mainly in terms of exposure and traction. Therefore, more research is required to enhance our ability to identify Fake News and combat its rising emergence in today's age.

Bibliography

[1] H. Allcott and M. Gentzkow. Social media and fake news in the 2016 election. *Journal of Economic Perspectives*, 31(2):211–36, May 2017.

[2] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. *Comput. Networks*, 29(8-13):1157–1166, 1997.

[3] M. Charikar. Similarity estimation techniques from rounding algorithms. In J. H. Reif, editor, *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 380–388. ACM, 2002.

[4] N. J. Conroy, V. L. Rubin, and Y. Chen. Automatic deception detection: Methods for finding fake news. In *Information Science with Impact: Research in and for the Community - Proceedings of the 78th ASIS&T Annual Meeting, ASIST 2015, St. Louis, Missouri, Missouri, USA, October 6-10, 2015*, volume 52 of *Proceedings of the Association for Information Science and Technology*, pages 1–4. Wiley, 2015.

[5] M. R. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In E. N. Efthimiadis, S. T. Dumais, D. Hawking, and K. Järvelin, editors, *SIGIR 2006: Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Seattle, Washington, USA, August 6-11, 2006*, pages 284–291. ACM, 2006.

[6] B. D. Horne and S. Adali. This just in: Fake news packs a lot in title, uses simpler, repetitive content in text body, more similar to satire than real news. *CoRR*, abs/1703.09398, 2017.

[7] T. Jansen and T. Kuhn. Extracting core claims from scientific articles. *CoRR*, abs/1707.07678, 2017.

[8] G. S. Manku, A. Jain, and A. D. Sarma. Detecting near-duplicates for web crawling. In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, editors,

Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007, pages 141–150. ACM, 2007.

[9] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky. The stanford corenlp natural language processing toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*, pages 55–60. The Association for Computer Linguistics, 2014.

[10] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[11] D. Paschalides, A. Kornilakis, C. Christodoulou, R. Andreou, G. Pallis, M. D. Dikaiakos, and E. P. Markatos. Check-it: A plugin for detecting and reducing the spread of fake news and misinformation on the web. In P. M. Barnaghi, G. Gottlob, Y. Manolopoulos, T. Tzouramanis, and A. Vakali, editors, *2019 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2019, Thessaloniki, Greece, October 14-17, 2019*, pages 298–302. ACM, 2019.

[12] V. Pérez-Rosas, B. Kleinberg, A. Lefevre, and R. Mihalcea. Automatic detection of fake news. pages 3391–3401, 2018.

[13] V. Rubin, N. Conroy, Y. Chen, and S. Cornwell. Fake news or truth? using satirical cues to detect potentially misleading news. In *Proceedings of the Second Workshop on Computational Approaches to Deception Detection*, pages 7–17, San Diego, California, June 2016. Association for Computational Linguistics.

[14] K. Shu, D. Mahudeswaran, S. Wang, D. Lee, and H. Liu. Fakenewsnet. `https://github.com/KaiDMML/FakeNewsNet`, 2018.

[15] K. Shu, D. Mahudeswaran, S. Wang, D. Lee, and H. Liu. Fakenewsnet: A data repository with news content, social context and dynamic information for studying fake news on social media. *CoRR*, abs/1809.01286, 2018.

[16] K. Shu, A. Sliva, S. Wang, J. Tang, and H. Liu. Fake news detection on social media: A data mining perspective. *ACM SIGKDD Explorations Newsletter*, 19(1):22–36, 2017.

[17] K. Shu, S. Wang, and H. Liu. Exploiting tri-relationship for fake news detection. *arXiv preprint arXiv:1712.07709*, 2017.

[18] J. Thorne and A. Vlachos. Automated fact checking: Task formulations, methods and future directions. pages 3346–3359, 2018.

[19] L. Wang, Y. Wang, G. de Melo, and G. Weikum. Five shades of untruth: Finer-grained classification of fake news. In U. Brandes, C. Reddy, and A. Tagarelli, editors, *IEEE/ACM 2018 International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2018, Barcelona, Spain, August 28-31, 2018*, pages 593–594. IEEE Computer Society, 2018.

[20] M. Zhang, X. Li, S. Yue, and L. Yang. An empirical study of textrank for keyword extraction. *IEEE Access*, 8:178849–178858, 2020.

Appendix A

A.1 Fake articles classified by domain published

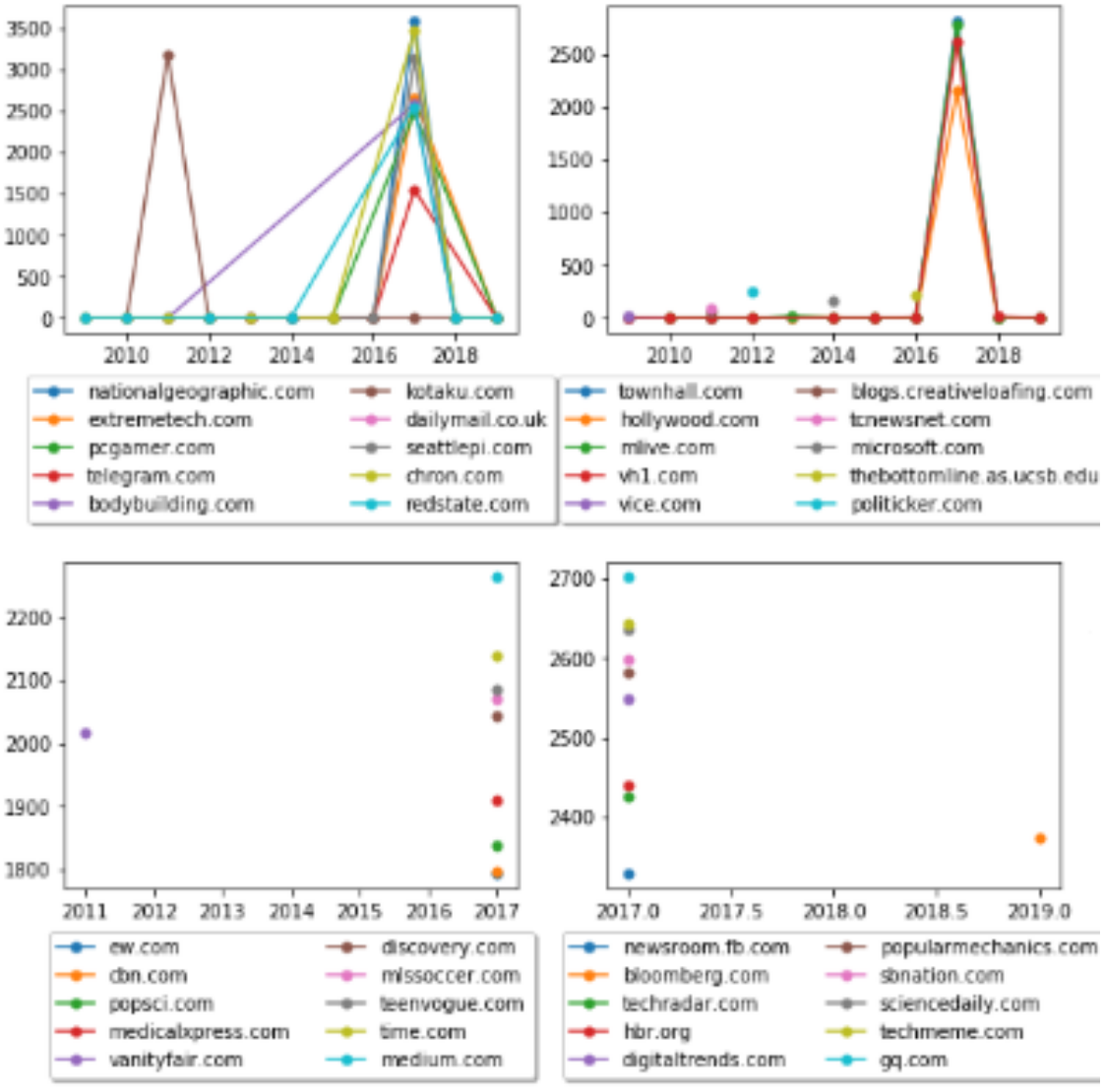


Figure A.1.1: Fake articles classified by domain published (Part 1)

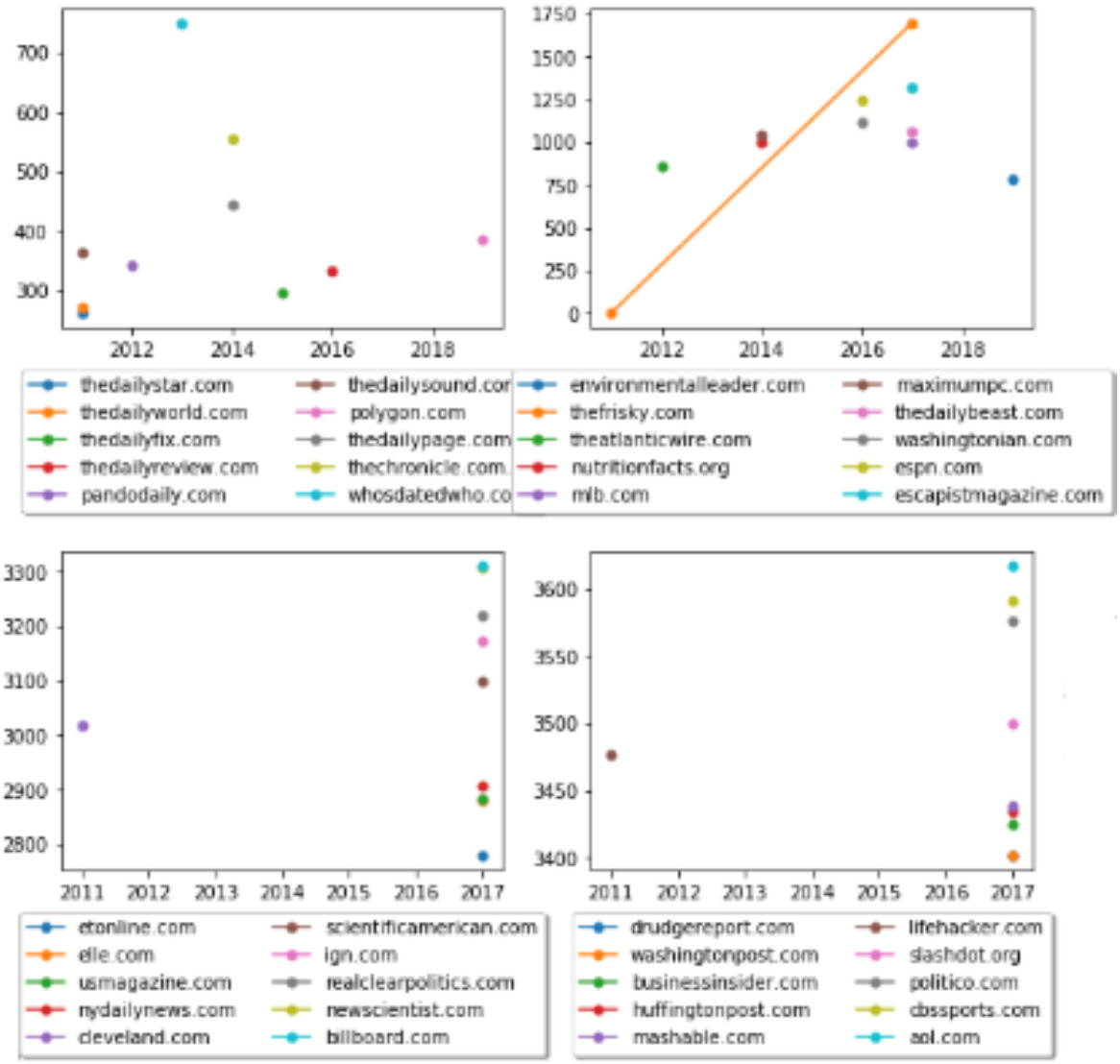


Figure A.1.2: Fake articles classified by domain published (Part 2)

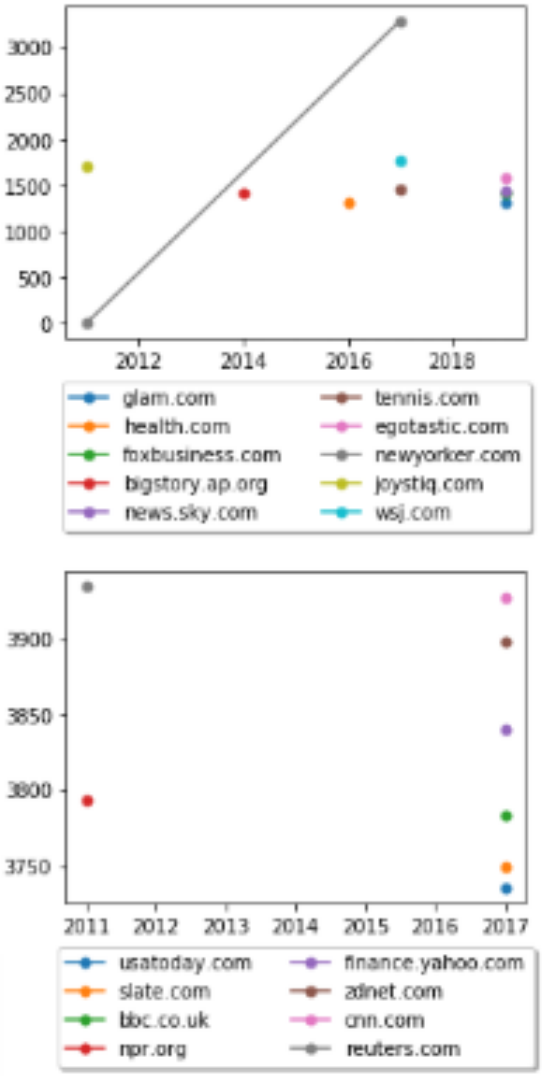


Figure A.1.3: Fake articles classified by domain published (Part 3)