

**FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO**

# **A Social Media Tool for Domain-Specific Information Retrieval - A Case Study in Human Trafficking**

**Tito Alexandre Trindade Griné**



Mestrado em Engenharia Informática e Computação

Supervisor: Carla Teixeira Lopes

July 18<sup>th</sup>, 2022



# **A Social Media Tool for Domain-Specific Information Retrieval - A Case Study in Human Trafficking**

**Tito Alexandre Trindade Griné**

Mestrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Prof. Maria Cristina de Carvalho Alves Ribeiro

External Examiner: Prof. Mayank Kejriwal

Supervisor: Prof. Carla Teixeira Lopes

---

July 18<sup>th</sup>, 2022



# Abstract

With the increasing growth of the Web and Social Media platforms, people are sharing more information than ever about themselves, using multiple websites with different purposes and thus leaving behind their digital footprint. Given the usefulness of this information, for example, in criminal investigations or journalism, there is an incentive to develop tools capable of accessing, linking, and storing this data.

However, since this data is generated without any concern for consistency or structure, depending on the website, it can be challenging to access and link it with other relevant information. Issues in using available search functionalities can arise, for example, when websites have search engines tailored to specific use cases but do not include many of the helpful query scenarios. Another situation is when the wanted information involves searching on multiple platforms. Finally, there may also be situations where websites lack any search functionalities or the existing ones are of significantly low quality.

In open-source intelligence tools, namely for Human Trafficking, the approach is usually centered around collecting large amounts of data from websites known to have correlations with Human Trafficking activity and then structuring it to facilitate data exploration, tag the data with indicators of criminal activity or directly classify it as containing or not suspicious signs of Human Trafficking. Due to the dynamic nature of the Web, these strategies can, over time, become less effective or even obsolete.

Relying on the assumption that most people reveal their presence on the Internet through sharing it on social media platforms and taking advantage of the fact that these platforms tend to expose good APIs, we propose a tool that extracts information scattered on the Web from people connected to a given domain by using Social Media profiles as the gateway to access this information, without relying on prior knowledge about where this information can be found.

For this, we begin by identifying profiles that are related to the latent domain contained in a set of keywords provided by the user. Resorting to topic modeling algorithms and word embeddings, the underlying domains of profiles can be extrapolated.

Following this, the selected profiles will be analyzed to extract links to other web pages that may exist in their descriptions or posts. These pages will be automatically analyzed to evaluate the degree to which they are related to the person or organization behind the profile.

Once both profiles and related websites have been collected, valuable information is collected and indexed to allow its rapid access and querying. An interface is also made available to configure and trigger searches and allow for the visualization and exploration of gathered results.

The tool was tested in a mock scenario related to Human Trafficking, specifically, sexual exploitation, by attempting to identify and collect information from Twitter profiles of people in the sex industry. The case study showed promising results, achieving an 87% precision in identifying profiles related to the domain and a 98% precision in identifying websites related to the profiles. This provides evidence that the approach is viable and that the tool has the potential to be used in real-world investigations to discover suspicious activity within the sex work industry.



# Resumo

Com o maior crescimento da Web e de plataformas de Redes Sociais, as pessoas partilham cada vez mais informação sobre elas próprias, ao utilizarem vários websites para diferentes propósitos e assim formando uma pegada digital. Dada a utilidade desta informação para, por exemplo, investigações criminais ou jornalismo, há um incentivo para desenvolver ferramentas capazes de aceder, interligar e guardar estes dados.

No entanto, uma vez que estes dados são gerados sem ter em mente a sua consistência ou estrutura, dependendo do website, estes dados podem ser difíceis de aceder e de interligar com informação relevante. Complicações em usar as funcionalidades de pesquisa disponíveis podem surgir quando, por exemplo, os websites têm motores de busca concebidos para casos específicos, não incluindo todas as possíveis utilizações úteis. Outro exemplo pode incluir situações em que se procura recolher informação existente em múltiplas plataformas. Por fim, existem também situações em que não existe a possibilidade de pesquisar informação através de um website, ou a pesquisa que existe retorna resultados de fraca qualidade.

Em ferramentas de investigação *open-source*, particularmente no combate ao tráfico humano, a abordagem centra-se em recolher largas quantidades de dados presentes em websites que se sabe terem evidências de atividade de tráfico humano, e estruturá-los para facilitar a exploração dos dados, classificá-los com indicadores de atividade criminal ou apenas identificar segmentos que contenham sinais suspeitos de tráfico humano. Devido ao dinamismo da Web, estas estratégias podem-se tornar menos eficazes ou até mesmo obsoletas.

Partindo do princípio que a maioria das pessoas revela a sua presença na Internet ao partilha-la em plataformas de Redes Sociais e tirando partido do facto de estas plataformas disponibilizarem APIs de boa qualidade, propomos uma ferramenta capaz de extrair informação dispersa pela Web, de pessoas intimamente ligadas a um domínio em particular, utilizando os perfis de Redes Sociais como a porta de acesso a esta informação, sem depender de conhecimento prévio relativamente a onde esta informação pode existir.

Para isso, começamos por identificar perfis que estão relacionados com o domínio latente contido num conjunto de palavras-chave, providenciadas pelo utilizador. Recorrendo a algoritmos de *topic modelling* e *word embeddings*, os domínios subjacentes aos perfis podem ser extrapolados.

De seguida, os perfis selecionados são analisados e são recolhidos *links* para outros websites que tenham sido partilhados nas descrições ou *posts* dos perfis. Estas *webpages* são então automaticamente analisadas para avaliar o grau de afinidade com o perfil do qual foram extraídas.

Uma vez recolhidos os perfis e *websites* relacionados, informação relevante é extraída e indexada para o seu rápido acesso e pesquisa. Uma interface é também disponibilizada para configurar e desencadear pesquisas, permitindo também a visualização e exploração dos dados recolhidos.

A ferramenta foi testada num cenário simulado relacionado com tráfico humano, especificamente de exploração sexual, para tentar identificar e recolher informação de perfis no Twitter de pessoas dentro da indústria do trabalho sexual. O caso de estudo mostrou resultados promissores, alcançando uma precisão de 87% na identificação de perfis relativos ao domínio, e uma precisão de

98% na identificação de *websites* relacionados com os perfis. Isto oferece provas que a abordagem é válida e que a ferramenta tem o potencial de ser usada em investigações reais na descoberta de actividade suspeita dentro da indústria do sexo.



# Acknowledgements

I would like to begin by sincerely thanking my supervisor, Carla Teixeira Lopes, for taking the risk of embarking on this project when it was just a hazy vision and enabling me to mature it into this dissertation. The consistent guidance and feedback wrapped in friendly conversations and good humor made the process much easier and more enjoyable. An extended thank you to INESC TEC for providing the resources that made it possible to better develop this work.

To my family and especially my parents, I want to give the utmost appreciation possible. Everything I have accomplished or will accomplish in life will always be thanks to the enormous support and love I receive from these two wonderful people. Mom and Dad, thank you for everything. I love you both immensely.

Lastly, in reaching the end of this journey through university, I must thank all my friends that made it truly special. To André, without whom I may have never found enjoyment in programming, thank you for all the fun moments and, of course, the enormous patience. To Cajó, the best company one could wish for, either on a long car ride back home or on a short walk to university, thank you for the many happy memories. To David, the gentle soul always by your side, thank you for the wonderful company and conversations even from a distance. To Manuel, the funniest personality I have ever met, thank you for all the challenging arguments and discussions that allowed me to grow as a person while still making it incredibly fun. And to all my other friends that have put a smile on my face in the past five years, thank you.

Tito Griné



*“Knowledge is freedom and ignorance is slavery”*

Miles Davis



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context . . . . .	1
1.2	Motivation . . . . .	2
1.3	Problem . . . . .	3
1.4	Goals . . . . .	3
1.5	Document Structure . . . . .	4
<b>2</b>	<b>Background</b>	<b>5</b>
2.1	Text Representation . . . . .	5
2.1.1	Term Frequency Inverse Document Frequency . . . . .	6
2.1.2	Bag-of-Words . . . . .	6
2.1.3	n-grams . . . . .	6
2.2	Similarity Metrics . . . . .	7
2.2.1	Jaccard Similarity . . . . .	7
2.2.2	Cosine Similarity . . . . .	7
2.3	String Matching . . . . .	8
2.3.1	Fuzzy Matching . . . . .	8
2.4	Parallel Computing . . . . .	9
2.5	Open-Source Intelligence . . . . .	10
<b>3</b>	<b>State of the Art</b>	<b>11</b>
3.1	Topic Modeling . . . . .	12
3.1.1	Topic Models . . . . .	12
3.1.2	Word Embeddings . . . . .	15
3.1.3	Applications of Topic Models for Web and Social Media-based content . . . . .	17
3.2	Forensic Technology for Human Trafficking . . . . .	19
3.2.1	Extensive Data Extraction and Linking . . . . .	20
3.2.2	Tagging Data with Human Trafficking Indicators . . . . .	22
3.2.3	Classification of Data as Suspicious Human Trafficking Activity . . . . .	26
<b>4</b>	<b>Problem Statement and Methodology</b>	<b>29</b>
4.1	Current Issues and Limitations . . . . .	29
4.2	Scope . . . . .	30
4.3	Assumptions . . . . .	31
4.4	Hypothesis . . . . .	31
4.5	Research Questions . . . . .	32
4.6	Architecture . . . . .	33
4.7	Process Flow . . . . .	33

<b>5</b>	<b>Implementation</b>	<b>37</b>
5.1	Profile Acquisition module . . . . .	37
5.2	Profile Selection module . . . . .	39
5.2.1	Process Flow . . . . .	39
5.2.2	Evaluation . . . . .	44
5.3	URL Extraction & Crawling module . . . . .	53
5.3.1	Process Flow . . . . .	54
5.3.2	Evaluation . . . . .	61
5.4	Data Storage and API . . . . .	66
5.4.1	Data Storage . . . . .	67
5.4.2	API . . . . .	68
5.5	Interface . . . . .	73
<b>6</b>	<b>Case Study in Human Trafficking</b>	<b>79</b>
6.1	Approach . . . . .	80
6.2	Results . . . . .	81
6.3	Conclusions . . . . .	83
<b>7</b>	<b>Conclusions</b>	<b>87</b>
7.1	Conclusions . . . . .	87
7.2	Challenges . . . . .	89
7.3	Contributions . . . . .	89
7.4	Future Work . . . . .	90
<b>A</b>	<b>Literature Review Process</b>	<b>93</b>
<b>B</b>	<b>Elasticsearch Profile Document Schema</b>	<b>95</b>
<b>C</b>	<b>Case Study Example Profiles</b>	<b>101</b>
	<b>References</b>	<b>105</b>

# List of Figures

3.1	PLSA model representation using plate notation . . . . .	13
3.2	LDA model representation using plate notation . . . . .	14
3.3	DIG architecture . . . . .	23
3.4	Three Level Framework . . . . .	24
3.5	Organized Crime Taxonomy (excerpt) . . . . .	25
4.1	Architecture Diagram . . . . .	34
4.2	Activity Diagram . . . . .	36
5.1	Profile Selection module's Classification phase diagram . . . . .	44
5.2	ROC curves for different word Embedding models . . . . .	47
5.3	ROC curves for the various profile components . . . . .	49
5.4	ROC curves for the different heuristics . . . . .	50
5.5	ROC curves for test and validation datasets (Profile Selection classifier) . . . . .	52
5.6	F1-score and accuracy as a function of the threshold value for test and validation datasets (Profile Selection classifier) . . . . .	53
5.7	Regular Expression formation diagram . . . . .	60
5.8	ROC curves for the initial search expressions . . . . .	63
5.9	ROC curves with and without tokens . . . . .	64
5.10	ROC curves for test and validation datasets (URL Extraction & Crawling classifier) . . . . .	65
5.11	F1-score and accuracy as a function of the threshold value for train and validation datasets (URL Extraction & Crawling classifier) . . . . .	66
5.12	Search Form Interface . . . . .	74
5.13	Searches Page Interface . . . . .	75
5.14	Search Configuration Modal Interface . . . . .	76
5.15	Search Results Page . . . . .	76
5.16	Profile Information Modal . . . . .	77
5.17	Profile Links Page . . . . .	77
5.18	Link Information Modal Interface . . . . .	78
C.1	Case Study Sex Worker Profile 1 . . . . .	101
C.2	Case Study Sex Worker Profile 2 . . . . .	102
C.3	Case Study Sex Industry Related Profile . . . . .	102
C.4	Case Study Unrelated Profile . . . . .	103





# List of Tables

3.1	Topic Modeling approaches for Web and Social Media-based content . . . . .	19
3.2	Principal Software Tools for Topic Modeling . . . . .	19
3.3	Human Trafficking Indicators . . . . .	27
5.1	Sources for list of people related to specific fields . . . . .	45
5.2	Distribution of the gathered profiles by topic . . . . .	45
5.3	Set of keywords used for each topic . . . . .	46
5.4	Characteristics of tested word embedding models . . . . .	47
5.5	AUC of tested word embedding models . . . . .	48
5.6	AUC for the different components . . . . .	49
5.7	AUC for the different heuristics . . . . .	50
5.8	AUC values for the different search expressions . . . . .	63
6.1	Case Study Searches Results . . . . .	82
6.2	Case Study Profiles Breakdown . . . . .	82
6.3	Case Study Related Links Data Breakdown . . . . .	83



# List of Listings

1	Searching configuration portion example . . . . .	38
2	Example query string for tweet searching . . . . .	38
3	Example word embedding operations . . . . .	43
4	Profile's ID schema . . . . .	67
5	Processed Links' Images schema . . . . .	68
6	Profile search query example . . . . .	71
7	Processed links search query example . . . . .	72
8	Example configuration for Case Study . . . . .	85
9	Profile data document schema . . . . .	99



# Abbreviations

API	Application Programming Interface
AUC	Area Under the Curve
BOW	Bag-of-Words
BTM	Biterm Topic Model
CNN	Convolutional Neural Network
DIG	Domain-specific Insight Graph
ESA	Explicit Semantic Analysis
FPR	False Positive Rate
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IR	Information Retrieval
JSON	JavaScript Object Notation
LDA	Latent Dirichlet Allocation
LSA	Latent Semantic Analysis
NCA	National Crime Agency
NER	Named-Entity Recognition
NGO	Non-Governmental Organization
NLP	Natural Language Processing
NN	Neural Network
OSINT	Open-Source Intelligence
PLSA	Probabilistic Latent Semantic Analysis
POS	Part-of-Speech
ROC	Receiver Operating Characteristic
SSE	Server-Sent Events
SPARQL	SPARQL Protocol and RDF Query Language
SVD	Singular Value Decomposition
SVM	Support Vector Machines
TCP	Transmission Control Protocol
TF-IDF	Term Frequency Inverse Document Frequency
TPR	True Positive Rate
UI	User Interface
UML	Unified Modeling Language
UNODC	United Nations Office on Drugs and Crime
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtual Machine



# Chapter 1

## Introduction

---

1.1	Context . . . . .	1
1.2	Motivation . . . . .	2
1.3	Problem . . . . .	3
1.4	Goals . . . . .	3
1.5	Document Structure . . . . .	4

---

This chapter introduces the problem behind this dissertation, starting by presenting the context in Section 1.1, followed by the motivation in Section 1.2. Section 1.3 explains the problem in more detail, which underpins the set of goals to achieve for this study, layed out in Section 1.4. Finally, in order to better guide the reader, Section 1.5 describes the structure of this document.

### 1.1 Context

The Web has been growing steadily in the last few years, meaning there is an ever-increasing number of users and information present on the Web. At the turn of the millennium, it was estimated that there were around 300 million users on the Web, roughly 5.0% of the world’s population. As of 2021, this number has increased to a staggering 5,000 million users, an equivalent to 65% of the world’s population [46].

With more people using the Internet, seemingly in more diverse ways, the more information they create and share, especially through Social Media platforms. As of January 2022, it is estimated that there are over 4,620 million social media users, which is about 58% of the world’s population [8]. Every person has an increasingly meaningful and more descriptive digital footprint every day [22].

Accessing this information and subsequently using it in productive ways can help answer important questions about our world. Although there is a risk of using this information in harmful ways, it is also possible to leverage it for social good. But, given that this data is generated by people on multiple platforms, there is an absence of structure and consistency. Both in academia

and the industry, much research is conducted on how to best access, link, store and analyze the relevant information. Nevertheless, with the dynamic nature of the Web, many challenges remain and old approaches become obsolete or less effective, carving a space for further innovation.

## 1.2 Motivation

People constantly interact with multiple platforms, ranging from commenting on a news article to creating an account on a social media platform and actively using it. Accessing information left by a person on various websites and cross analyzing it can be instrumental in a multitude of scenarios, but doing so effectively and efficiently is challenging.

The first issue is with websites that have search functionalities that are tailored to a specific use case and, as such, don't cover all potentially helpful scenarios. To exemplify this, we can consider a website like GitHub that has an excellent search engine for finding code and repositories. This covers the most prominent use case of the platform, storing and sharing code. Nevertheless, we can imagine a company interested in looking up profiles from users interested or experts in a particular field. Using the native search functionalities, there is no clear and effective way of doing this.

Another issue can arise when the desired information is scattered across many places. Take a newspaper that wants to find articles published by journalists experienced in a given field. The articles could be present in multiple outlets, from blogs to digital newspapers.

The last and more extreme version of the problem is when websites and platforms do not offer any search options over the data they keep, or these are of significantly low quality. This could be not having a publicly available API to access its data or not supporting manual searching. Such could be the case for applications that do not have the resources or technical scale to offer good search capabilities or do not want bots to be able to query their system.

To illustrate this last case, one can take the example of the OnlyFans social network. OnlyFans is a platform to share and directly monetize user-created content. Although not conceived explicitly for this purpose, the website has risen to popularity in the past few years as a means to offer sexually explicit content. Due to its minimal regulation and bear to entry, it has been increasingly tied to cases of sexual exploitation of minors and instances of human trafficking [39]. Authorities and NGOs interested in conducting investigations on this platform to prevent its use for these purposes need to search it both for leads on existing cases or finding new cases. Perhaps intentionally, OnlyFans has poor search functionalities, not only because they do not expose an API to the public, but also because searching manually returns only a handful of low-quality results.

Designing a tool able to circumvent these problems, even if partially, could constitute a vital resource to aid investigations in finding leads, organizations and human-resources personnel to discover apt candidates for a position, journalists in selecting relevant people to interview. Generally, it could help people in efficiently and effectively use the Web as a source of valuable and useful information.



## 1.3 Problem

Websites vary significantly in their search functionalities. Some do not offer any, others offer low-quality ones, while some cover a portion but not all meaningful search necessities. This means that accessing and linking information from the Internet is not trivial, but its usefulness in investigations, journalism, and other fields remains.

Social Media platforms are arguably where most people use the Internet and share information [45]. There are as many social media users as 93.4% of all internet users, and it is reported that, on average, 2 hours and 27 minutes are spent per day on social media by each user [8]. Among the twenty most used social platforms is Twitter with 436 million users. It is estimated that about 5.5% of people on Earth use it [9].

Being well established in the internet space, these social media platforms often expose good quality APIs to the public, albeit with usage restrictions.

Under the assumption that people tend to share their presence on the Internet through social media and that the larger this presence is, the more likely it is that it is accessible through social media, this research proposes a tool using social media APIs as a gateway to access more information contained on the Web. It differentiates itself from other data extraction and linking technologies, like Google Search, by relying on Social Media platforms and their users to point to the relevant information. Instead of focusing on finding documents related to a particular query, the focus is instead on finding social media profiles that hopefully correlate to real-world entities, sufficiently related to the themes of said query, and then branch out looking for related information associated with those profiles. Some of this information will be useful in answering queries regarding the topic and the people or organizations related to it.

## 1.4 Goals

The main goal of this dissertation is the building of a tool to extract information scattered on the Web from people related to a given domain using Social Media as a gateway. The tool will retrieve social media profiles related to the topic or topics present in a set of given keywords, as well as information gathered from those profiles with a special emphasis on links to other relevant websites and the corresponding information retrieved therein. The gathered information is to be appropriately stored and indexed, which, when combined with an appropriate interface, will allow for its exploration and querying. The tool is to be performant, scalable, and domain-agnostic.

Furthermore, a secondary goal is to demonstrate the applicability and usefulness of the developed tool as Open-Source Intelligence (OSINT) for investigative settings by using it in the Human Trafficking and sexual exploitation domain. The hope is that this demonstration leads to its test and eventual adoption by law enforcement and non-governmental organizations (NGOs) to support their efforts to combat these issues.

Lastly, to extend its reach and usefulness, the source code for the tool is to be made available as open-source, where it will hopefully be applied to other cases in various domains.

## 1.5 Document Structure

Following this Introduction, Chapter 1 (p. 1), the rest of the document contains another six chapters with the following structure:

- **Background**, Chapter 2 (p. 5), describes important concepts used throughout the work, where a good understanding helps to better conceptualize the project;
- **State of the Art**, Chapter 3 (p. 11), presents current technologies relevant for this dissertation and applications thereof in the domain of web and social media content;
- **Problem Statement and Methodology**, Chapter 4 (p. 29), details the problem tackled, the adopted methodology, the developed solution and its overall functioning;
- **Implementation**, Chapter 5 (p. 37), walks through each major component of the application and describes its inner workings, justifications behind some design decisions and the tests performed to evaluate the performance of certain modules;
- **Case Study in Human Trafficking**, Chapter 6 (p. 79), frames the use of the developed tool in the domain of Human Trafficking, more specifically of sexual exploitation, and recounts the case study performed in this domain that showcases the usefulness of the tool in related investigations;
- **Conclusions**, Chapter 7 (p. 87), summarizes the main accomplishments and takeaways from the research and development work, and postulates improvements that could be done in the future.

## Chapter 2

# Background

---

<b>2.1</b>	<b>Text Representation</b>	<b>5</b>
<b>2.2</b>	<b>Similarity Metrics</b>	<b>7</b>
<b>2.3</b>	<b>String Matching</b>	<b>8</b>
<b>2.4</b>	<b>Parallel Computing</b>	<b>9</b>
<b>2.5</b>	<b>Open-Source Intelligence</b>	<b>10</b>

---

This chapter serves to introduce and establish some key concepts that will be useful to better comprehend and follow the rest of this work. More specifically, it will begin with the introduction to standard schemas for text representation in Section 2.1, namely the Term Frequency Inverse Document Frequency model, the Bag-of-Words schema, and n-grams. It follows with the description of two popular similarity metrics, Jaccard Similarity and Cosine Similarity, in Section 2.2. A brief explanation of the principal approaches to string matching is done in Section 2.3, with a particular emphasis on Fuzzy Matching. It is followed by an introduction to parallel computing based on a description of the Map-Reduce paradigm in Section 2.4. Finally, Section 2.5 presents an overview of Open-Source Intelligence.

### 2.1 Text Representation

When dealing with text and especially with large corpora, there is a need to represent it in such a way that it is practical for algorithms to manipulate it but also maintains the useful properties or characteristics of the original text. Such representations are used in many fields of computer science, from information retrieval, natural-language processing, machine learning, among others.

This section presents three common and widely used strategies for representing text: the TF-IDF vector approach, Bag-of-Words schema, and n-grams. All can be used as a vector space model, which is a model for representing text documents as vectors [48].

### 2.1.1 Term Frequency Inverse Document Frequency

The *Term Frequency-Inverse Document Frequency* (TF-IDF) can be defined as the product of the relative frequency of words in a specific document by the inverse proportion of those words over the entire document corpus [43]. Essentially, it tells us how relevant a given term is in a specific document. Common terms like articles and propositions tend to have low TF-IDF values, but terms that only appear in a small subset of documents will have high TF-IDF values in those documents.

Calculating the TF-IDF value of a term  $t$  in a document  $d$ , taken from the document corpus  $D$  is done by

$$tf-idf_{t,d} = tf_{t,d} \times \log\left(\frac{|D|}{tf_{t,D}}\right) \quad (2.1)$$

where  $tf_{t,d}$  is the number of times the term  $t$  appears in  $d$ ,  $|D|$  is the size of the document collection and  $tf_{t,D}$  is the number of times term  $t$  appears in  $D$  [47].

We can divide the equation in two. Where  $tf_{t,d}$  represents the Term Frequency (TF), that is, the relative frequency of the term  $t$  within document  $d$ . And  $\log\left(\frac{|D|}{tf_{t,D}}\right)$  is the Inverse Document Frequency (IDF), a measure of the amount of information the term  $t$  provides.

Using TF-IDF, text can be represented as a vector of the respective TF-IDF values of each word present in the document, all others being 0. If the text is isolated, that is, it doesn't come from a larger, related corpus, the IDF values can be calculated with respect to a corpus of relevant documents, for example, all of Wikipedia's articles.

### 2.1.2 Bag-of-Words

The *Bag-of-Words* (BOW) schema is a simple yet extensively used approach for representing text. A BOW representation of a text is a list of the number of occurrences of all the words in a corpus [14]. In its original definition, this sort of representation discounts grammar and word order, keeping only the word incidence.

### 2.1.3 n-grams

In its most general form, an *n-gram* is just a continuous sequence of  $n$  items from a given text. The items can range from characters, syllables, words, or other meaningful segmentation of text. When referring to word  $n$ -grams, one can use the term *shingles* [6].

As mentioned, the standard BOW representation does not account for word order, which, albeit irrelevant in some cases, is not always desired. As such, when these properties are wanted, a Bag-of- $n$ -grams representation can be used to preserve these characteristics. As an example, a word-based Bag-of-2-grams representation is a vector of the number of co-occurrences of any two words existing in the corpus, that is, any two words found together in the text. This representation preserves some of the original word order present in the text.

## 2.2 Similarity Metrics

In Section 2.1, different strategies for representing text were presented. These representations facilitate operations on text, and a common goal when having two texts is to discover how similar they are to each other. The answer to this question largely depends on what is considered “similar”. For example, in a lexical sense, the similarity may be how many characters or words co-exist in relation to the total. In a semantic sense, the similarity may be given by how related the topics from the two documents are.

Interestingly, the meaning of similarity is more contingent on how the text is represented, that is, what features are present in its representation, than on the specific manner the similarity is calculated, albeit different similarity metrics can be more or less well suited for particular representations.

As such, leaving aside the actual meaning of similarity, a *similarity metric* or *similarity measure* is a function that takes in two objects and outputs a real-value that represents the similarity between them. The rest of this section is dedicated to defining two of the most popular similarity measures used, namely Jaccard similarity and cosine similarity.

### 2.2.1 Jaccard Similarity

The *Jaccard similarity* or *Jaccard coefficient* is usually used to compare the similarity of two sets and is defined as the size of the intersection divided by the size of the union [29]. In mathematical notation, considering sets  $A$  and  $B$ , it can be written as:

$$\frac{|A \cap B|}{|A \cup B|} \quad (2.2)$$

For text representations, it can be used to calculate the similarity between two documents represented as BOW, see Section 2.1.2, for example, but does not take into account the frequency of each word.

### 2.2.2 Cosine Similarity

The *cosine similarity* applies to any two non-zero vectors and is defined as the cosine of the angle between them [50]. This implies that the cosine similarity is a value between  $[-1, 1]$ . In mathematical notation, considering vectors  $A$  and  $B$  with  $\theta$  being the angle between them, it can be written as:

$$\cos(\theta) = \frac{A \cdot B}{||A|| \times ||B||} \quad (2.3)$$

A significant advantage of this metric is that it can be used with any real-value vectors of any dimension. For text representations, since they tend to be sparse, that is, be vectors mainly consisting of zeros, it has the added benefit of only considering the non-zero values for the calculation.

Applying cosine similarity to TF-IDF vectors, see Section 2.1.1, is very common as it tends to be a good predictor of similarity even when the two documents have significantly different lengths.

## 2.3 String Matching

A *Regular Expression* or *regex* is a character sequence that defines a search pattern in text. These expressions are widely used in computer science to perform “find operations” on text, that is, to locate strings in a body of text that match a particular pattern [56].

A regular expression can be as simple as a string where each character is to be interpreted literally. In this case, the pattern to search for is the literal regex string. Special characters can be used to extend the search pattern by, for example, including wildcard characters or allowing some set of characters to repeat up to a given number of times. With these instruction characters, regular expressions become significantly more useful as they allow the matching of strings without explicitly enumerating every possible combination, which in many situations would be impossible. For example, the Python regex, `/([A-Za-z0-9]+[._-])*[A-Za-z0-9]+@gmail.com/`, matches all valid Gmail addresses by giving instructions on the characters the username can have.

### 2.3.1 Fuzzy Matching

Albeit extremely useful for many scenarios, standard regular expressions only look for exact matches of a particular search pattern. Some situations, however, require *approximate string matching*, often called *fuzzy string matching*, that is, the search for strings that approximately match a pattern. For it to be possible, it requires a definition of what exactly is an approximate match since, in the most extreme interpretation, any string can be an approximate match of any other string.

In fuzzy matching, the notion of an approximate match is closely linked to the concept of the Levenshtein distance. The *Levenshtein distance*, introduced by Vladimir Levenshtein, is a metric for measuring the distance or difference between two strings or sequence of characters [28]. Broadly, the distance can be defined as the minimum number of single-character edits needed to transform one string into the other. Edits can include any single-character insertion, deletion, or substitution. The following equation gives a formal, recursive definition of the Levenshtein distance:

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

Where  $|x|$  is the length of the string  $x$ ,  $x[n]$  is the character of string  $x$  at position  $n$ , and  $\text{tail}(x)$  is the string  $x$  without the first character.

When writing a regular expression for fuzzy matching, a limit is defined for the pattern or subsets of the pattern on the maximum Levenshtein distance allowed between an exact match string and an approximate match string. For example, setting a maximum distance of two means that, at most, the returned match string has to be two edits away from an exact match.

Fuzzy matching provides an excellent mechanism for searching for terms in text that can have misspellings or slight variations, but what differences exist exactly are unknown or require enumerating all possibilities in a complex regular expression.

## 2.4 Parallel Computing

When dealing with large amounts of data, performing computations in a standard linear fashion quickly becomes untenable. Taking advantage of the fact that in many situations, operations are independent of each other, that is, the result of a computation on a given piece of data doesn't influence the result of another computation on a different data segment, programming systems are designed for parallelism, so multiple computations can happen "simultaneously". This can be achieved through multiple CPU cores, or through "*computing clusters*", which are large collections of commodity hardware connected by Ethernet cables [27]. One possible approach to this sort of parallelism relies on two principal concepts.

The first is a "*Distributed File System*", which is essentially a storage file system that uses multiple host machines connected through a computer network to store files and make them accessible to multiple computers. It is essential because it allows the ensemble of "*compute nodes*" from a computing cluster to access the same resources.

The second is the "*MapReduce*" programming paradigm. In essence, it is a style of computing, initially introduced by Google's MapReduce, to program large-scale parallel computations in a hardware fault-tolerant manner. The programmer only needs to define two functions **Map** and **Reduce**, while the system deals with managing the execution and coordination of the Map and Reduce tasks, as well as possible failures that can arise. The MapReduce computations are executed in the following manner [27]:

1. Chunks from a distributed file system are distributed amongst a number of Map tasks. These tasks turn each chunk into a sequence of *key-value* pairs using the user-defined Map function;
2. A *master controller* collects and sorts all *key-value* pairs and then splits them across all the Reduce tasks, such that all pairs with the same key are mapped to the same Reduce task;
3. Every Reduce task takes one key at a time and, using the user-defined Reduce function, combines all the values associated with that key.

In short, the programmer's main concern is to simply define Map and Reduce functions, being that Map functions take as arguments key-value pairs and are supposed to return zero or more key-value pairs, and Reduce functions receive a key and a list of values associated with that key, subsequently returning zero or more values. Some situations may not even require the Reduce portion the MapReduce paradigm. Even so, the concept of distributing a dataset and map or parallelize computations on that dataset is immensely useful for allowing the processing of large amounts of data without having the computation time increase proportionally.

## 2.5 Open-Source Intelligence

The tool that serves as the proof-of-concept for this dissertation's proposal falls under the category of Open-Source Intelligence, so it is important to know what exactly it constitutes. Steele [51] defines *Open-Source Intelligence* as:

*“Open source intelligence, or OSINT, is unclassified information that has been deliberately discovered, discriminated, distilled, and disseminated to a select audience in order to address a specific question.”*

In other words, it is the collection and analysis of publicly available data to gain valuable insights and help develop solutions to real-world problems.

Best [3] breaks down the Open Source Intelligence cycle into five technical processes:

- **Collection**, includes information retrieval through searching and/or crawling;
- **Process**, pertains to information extraction, such as extracting and classifying entities, places, and the corresponding links/relationships between them;
- **Analyse**, refers to analyzing the processed data such as its links, relationships looking for trends or patterns, and collecting statistics;
- **Visualize**, is concerned with visualizing in a meaningful way the data and its findings, through networks, graphs, maps, and other tools;
- **Collaboration**, deals with using the insights and methods combined with existing OSINT data and tools.



## Chapter 3

# State of the Art

---

3.1 Topic Modeling . . . . .	12
3.2 Forensic Technology for Human Trafficking . . . . .	19

---

Following the previous chapter where key concepts were introduced, this chapter encapsulates the state of the art of some relevant research areas for this dissertation. The study of the current state of the art was done through an iterative review process of the existing literature. For a systematized description of the process see Appendix A (p. 93).

The process began by defining possible research fields pertinent for this project. To select relevant keywords, a quick analysis of the corresponding field's Wikipedia page was done, since it usually comprises a brief overview. This is useful to establish a baseline of commonly used vocabulary within the field.

A search query was then defined by concatenating the keywords with specific search engine operators (*e.g.*, AND, OR). This query was then used to retrieve articles. Results were selected based on their publishing date, number of citations and relevance to the domain. This relevance was determined by a combination of the title and abstract.

From the selected articles and taking a *snow-balling* approach, more literature was considered by reviewing the cited works through a process similar to the one described above.

The Google Scholar<sup>1</sup> web search engine was chosen for querying and subsequently retrieving the relevant literature from the many digital libraries available. Most of the literature was available on ACM Digital Library<sup>2</sup>, IEEE Xplore<sup>3</sup>, Springer Link<sup>4</sup>, ASIST&T<sup>5</sup>, Research Gate<sup>6</sup>, Science Direct<sup>7</sup>. The literature includes journal articles, conference papers, books, reports, news and web articles.

---

<sup>1</sup>Google Scholar - [scholar.google.com](https://scholar.google.com)

<sup>2</sup>ACM Digital Library - [dl.acm.org](https://dl.acm.org)

<sup>3</sup>IEEE Xplore - [ieeexplore.ieee.org](https://ieeexplore.ieee.org)

<sup>4</sup>Springer Link - [link.springer.com](https://link.springer.com)

<sup>5</sup>ASIST&T - [www.asist.org](https://www.asist.org)

<sup>6</sup>Research Gate - [www.researchgate.net](https://www.researchgate.net)

<sup>7</sup>Science Direct - [www.sciencedirect.com](https://www.sciencedirect.com)

The state of the art follows with Section 3.1, that provides a thorough overview of the history and recent applications of Topic Modeling with a special regard for social media-based content. Finally, Section 3.2 discusses the solutions devised in recent years to help investigations related to Human Trafficking, serving as forensic technology.

## 3.1 Topic Modeling

Topic modeling aims to discover latent or abstract topics from a collection of text documents, a tool widely used in text mining. A topic model is defined as a statistical model capable of extracting these hidden topics and, therefore, some of the underlying semantics from a given text. The general assumption is that every text concerns one or more topics, and so, words related to the topics are more likely to appear in the document [57].

Section 3.1.1 will start by covering the initial and more well-known topic models developed as early as 1990. Following this, Section 3.1.2 will go over different algorithms for obtaining word embeddings that preserve semantic meaning and, as such, have interesting applications in topic modeling. Lastly, Section 3.1.3 explores the uses and variations of these models, some combined with word embeddings, for topic modeling of web and social media content.

### 3.1.1 Topic Models

Early works of topic modeling stemmed from a need to improve the existing text indexing and retrieval methods [10].

Given a user query  $q$ , the goal was to retrieve the data that best matched the query, going beyond just simple word matching, and instead discover the underlying topics the user was interested in getting, going by their choice of words in  $q$ .

Deerwester *et al.* [10] identified two problems that common word matching algorithms did not manage to tackle. First is the fact that multiple words can be used to describe the same concept, referred to as the *synonym* problem. Second, the same word can have different meanings depending on the context in which it is being used, the *polysemy* problem.

In an attempt to address these issues, the authors proposed the Latent Semantic Analysis (LSA) model. It starts with an arbitrary rectangular matrix of terms and documents. The matrix's Singular Value Decomposition (SVD) is calculated, resulting in three matrices that break down the original relationships between the terms and the documents into linearly independent factors. Some of these factors will be small and thus negligible. If ignored, an approximate model with fewer dimensions than the original can be obtained that still contains the document-document, term-term, and document-term similarities. This result can be viewed as a geometric spatial representation of the latent semantic space, where positions in this space function as a sort of semantic indexing. Furthermore, distances in this space, calculated for example, through cosine distance, can be viewed as a measurement of semantic similarity between documents.

In short, the rationale behind the model is that similarities between documents can be better estimated in the low dimensional space obtained through SVD. In this latent space, the assumption

is that documents that contain frequently co-occurring terms will have similar representations, even if they have no terms in common. That is, if document  $A$  has the term  $a$ , document  $B$  has the term  $b$ , and both  $a$  and  $b$  are frequently co-occurring terms, then documents  $A$  and  $B$  may have similar representations, even though they do not share any terms.

Hoffman [17] considered that LSA did not have a robust statistical foundation and proposed a new model, the Probabilistic Latent Semantic Analysis (PLSA) model, which, at its core, uses the statistical model called the *aspect model* [16]. It associates an unobserved class variable, in this case, the latent topic, to each observation, in here represented as a word-document co-occurrence pair.

The assumption is that each text document contains multiple latent topics, and each word in the text relates to a particular topic. The probability of a word being in a given document can be obtained by summing, for every topic, the product of the probability of that word appearing given that topic and the probability of that topic being present given the document. Considering  $d$  as a document label,  $z$  is a topic, and  $w$  as a word, the probability can be written in mathematical notation as:

$$P(w|d) = \sum_z P(w|z)P(z|d) \quad (3.1)$$

Through various calculations that are beyond the scope of this project, the author shows that the model can converge on a local maximum and attach a probability that a topic is present given the document.

In short, the PLSA models each word in a document as a sample from a mixture model, where each component is a multinomial random variable interpreted as a topic. Consider Figure 3.1, where  $d$  represents an observed document,  $w$  is an observed word and  $z$  is a hidden topic. The model assumes that each document draws from its topics a word, and each word has a certain probability of being generated from a single topic.

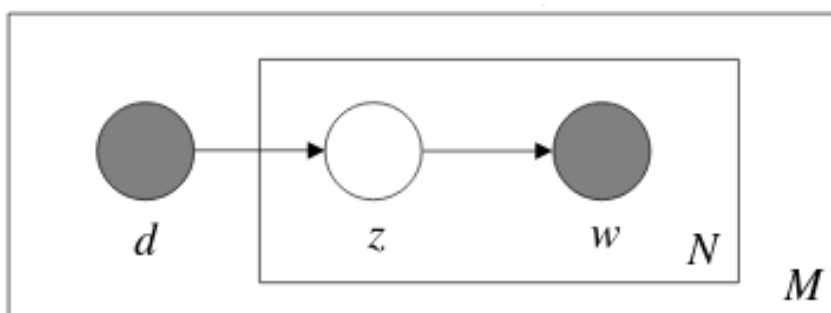


Figure 3.1: PLSA model representation using plate notation taken from Blei *et al.* [4].

Blei *et al.* [4] point out some issues with the PLSA model, namely that:

- The number of parameters the model has to train grows linearly with the model's size. This results in the model's severe propensity to overfit;

- It is unclear as to how one assigns probabilities to a document outside of the training dataset;

To address these issues, the authors propose what is arguably the most used topic model currently, the Latent Dirichlet Allocation (LDA) model. The authors describe the model as follows:

*“LDA is a three-level hierarchical Bayesian model in which each item of a collection is modeled as a finite mixture over an underlying set of topics. Each topic is, in turn, modeled as an infinite mixture over an underlying set of topic probabilities. In the context of text modeling, the topic probabilities provide an explicit representation of a document.”*

In other words, LDA is a generative probabilistic model that considers that each document can be seen as a random mixture of latent topics, each one characterized by a distribution over words. The algorithm aims to find the best probability distributions of both document-topic and topic-word representations.

Consider Figure 3.2, LDA defines the following generative process, for each document in a corpus:

1. For each document, pick a topic from its topic distribution  $\theta$ ;
2. Sample a word  $w$  from the word distribution associated with the chosen topic  $z$ ;
3. Repeat the process for every word in the document.

The parameters  $\alpha$  and  $\beta$  account for the Dirichlet priors on the per-document topic distributions and per-topic word distributions, respectively.

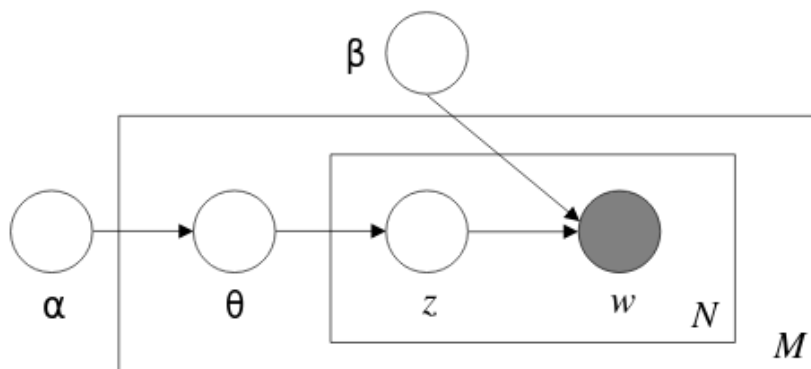


Figure 3.2: LDA model representation using plate notation taken from Blei *et al.* [4].

It's important to point out that, unlike LSA and PLSA, which relied on the Bag-of-Words assumption that the order of the words in a document can be ignored, LDA can be applied to any n-gram structural unit and therefore consider word order.

An extension of the LDA model was proposed by Rosen-Zvi *et al.* [44] to consider documents generated by the same author. The Author-Topic model assumes that each word is associated with two latent variables, the author and the topic, where an author is represented as a multinomial distribution of topics. For this model, the observed variables are both the set of words as well as the set of authors.

Concerned with the application of the standard models in short corpora, Yan *et al.* [60] proposed the Biterm Topic Model (BTM), which addresses the issue of data sparsity when dealing with small text. The authors point out that, when dealing with short texts, directly applying the standard models is not effective since:

- Occurrences of words play a less discriminative role given that the model does not have enough word counts to know how words are related;
- Limited context increases the difficulty in identifying the senses of ambiguous words.

The BTM uses biterms, which are unordered word pairs co-occurred, and explicitly models the word co-occurrence patterns rather than documents. So the generative process can be described as first picking a topic and then extracting word pairs from that topic's distribution for each word pair in a document.

One important distinction is that BTM does not directly obtain the topic proportions of documents during the learning process. However, these can be inferred by assuming that the topic proportion is equal to the expectation of the topic proportions of biterms generated from that document. Considering  $d$  as a document label,  $z$  is a topic, and  $b$  is a biterm, the probability can be written in mathematical notation as:

$$P(z|d) = \sum P(z|b)P(b|d) \quad (3.2)$$

The authors tested their model on a dataset of Tweets and normal-sized text and found that it outperformed standard models in both scenarios.

It is relevant also to mention the work of Gabrilovich *et al.* [12] that introduced the Explicit Semantic Analysis (ESA) model. The model relies on a considerable text corpus, in the paper, Wikipedia is used as a knowledge base. It represents every word as a weighted vector of concepts, each concept here is related to a given Wikipedia article. The weight of a given word is calculated using the TF-IDF scheme. The combination of all word vectors is named the weighted inverted index. This index can then map a given text to the concept space defined by the knowledge base. After this mapping, the vector representation of text can be used to compare two documents, or the similarity of a document to a given concept, using distance metrics like cosine distance.

### 3.1.2 Word Embeddings

Statistical models for topic modeling effectively attempt to find a representation of a text as some combination of various topics related to the present words.

In LSA, it is the SVD decomposition of the term-document matrix and the projection of text onto the resulting lower-dimensional space. In PLSA and LDA, it is the probability distribution of the latent topics found. In either case, comparing two documents or a document and a given topic can be understood as the distance between their representations in the spaces created by the topic models.

A word embedding, in NLP, is the representation of words for text analysis, usually in the form of vectors of real numbers, such that the meaning of a word is encoded in its representation. As such, words that are closer in the vector space are expected to be similar in meaning [21].

If appropriate word embeddings are obtained, comparing two documents can be done, for example, by the cosine distance of their vector representations, which is itself obtained by the concatenation of the word embeddings of all words that make up each document.

Mikolov *et al.* [34] presents two Neural Network (NN) models capable of generating such word embeddings by being trained on a large text corpus through unsupervised learning. As mentioned by the authors, word representations obtained through LSA are not great at preserving linear regularities among words. Using LDA also has the adverse effect of becoming computationally expensive when training on large datasets. The two models that are described are:

- **Continuous Bag-of-Words Model**, which is trained to predict the next word likely to appear based on the input words that surround it;
- **Continuous Skip-gram Model**, which is trained to predict words in a given range before and after an input word;

The logic behind these models is that, by being trained to predict words based on words close to it, the hidden layer must become an embedding layer for words. Since similar words will appear in similar contexts, their representation in this layer should evolve to be similar as they will output nearly identical results.

The algorithm was made available and popularized through the word2vec<sup>8</sup> tool.

Another model using unsupervised learning was introduced by Pennington *et al.* [41] called GloVe<sup>9</sup>.

The authors made the distinction of two model types, each with its strengths and deficiencies:

- **Global Matrix Factorization methods**, where LSA and LDA are included, leverage statistical information from the whole document, but their vector space structure tends to not convey word analogies, such as, “*king is to queen as man is to woman*”.
- **Local Context Window methods**, which include the aforementioned NN models, although good at analogy tasks, rely only on local context and not on the global co-occurrence counts.

The put forth model attempts to combine the features of both models. The underlying idea behind the GloVe algorithm is that ratios of co-occurrences probabilities are a better starting point

<sup>8</sup>word2vec - [www.code.google.com/archive/p/word2vec/](http://www.code.google.com/archive/p/word2vec/)

<sup>9</sup>GloVe - [nlp.stanford.edu/projects/glove/](http://nlp.stanford.edu/projects/glove/)

for training than the probabilities themselves. That is, the relationship between two words can be more accurately mapped not by their word-word co-occurrence probability, but by the ratio of their co-occurrence probabilities with another word, named the context word.

Finally, the current state-of-the-art in multiple NLP tasks is credited to the BERT model for language representation, introduced by Devlin *et al.* [11]. The model is based on the transformer architecture for NN, proposed by Vaswani *et al.* [54], which is based solely on attention mechanisms. BERT is first trained on unlabeled data in the pre-training phase. The model's parameters are then used to initialize another model of very similar architecture and subsequently fine-tuned on a variety of downstream tasks, that is, the actual tasks the model is intended to solve.

### 3.1.3 Applications of Topic Models for Web and Social Media-based content

The idea of using topic models to analyze and extract meaning from text found on the web, particularly on social media websites, has been extensively explored in the literature.

Onan [40] designed an improved word embedding scheme-based representation using a combination of different word vector representations, such as word2vec [34] and LDA2vec [35], which is a combination of dense word vectors with LDA [4], to make the obtained representation more human interpretable. This scheme was used in conjunction with a clustering ensemble in order to extract topics from Bibliometric Data obtained by crawling the Web of Science (WoS) database.

Uteuov *et al.* [53] combined probabilistic models, like LDA [4] and ARTM [55], with NN-based word representations, such as word2vec [34], in a model called deARTM which the authors claim obtains relationships between documents encompassing various abstraction levels, each with its own vector representation. They then tested their model on various datasets, including text extracted from social media platforms like VK <sup>10</sup> and Facebook <sup>11</sup>.

Focusing their attention on short text, evermore prevalent on social media, Qiang *et al.* [42] took the Markov Random Field Regularized (MRF) [58] model and incorporated it with word embeddings taken from word2vec [34] and GloVe [41]. The resulting model called the Embedding-based Topic Model (ETM) was then tested on short text datasets with data taken from Twitter <sup>12</sup> and GoogleNews <sup>13</sup>.

Using the eRisk 2018 dataset <sup>14</sup> containing writings taken from users on the Reddit <sup>15</sup> social media platform, Maupomé *et al.* [32] applied the LDA topic model [4] to extract discussion topics which were then used to train a Multilayer Perceptron to assess the risk of depression.

An LDA topic model [4] was also used by Liu *et al.* [30] to extract topics from Instagram <sup>16</sup> comments and posts from the Helsinki region and identify city events. Their approach also used

---

<sup>10</sup>VK - [www.vk.com](http://www.vk.com)

<sup>11</sup>Facebook - [www.facebook.com](http://www.facebook.com)

<sup>12</sup>Twitter - [www.twitter.com](http://www.twitter.com)

<sup>13</sup>Google News - [www.news.google.com](http://www.news.google.com)

<sup>14</sup>eRisk 2018 - [www.early.irlab.org/2018/index.html](http://www.early.irlab.org/2018/index.html)

<sup>15</sup>Reddit - [www.reddit.com](http://www.reddit.com)

<sup>16</sup>Instagram - [www.instagram.com](http://www.instagram.com)

word embeddings taken from word2vec [34] to gather seed words related to the input words and use them to retrieve Instagram posts.

Focusing on Twitter, Xu *et al.* [59] extended the Author-Topic Model [60] to discover Twitter users' interests. Their modified model attempts to address the problem of users' tweets not always being related to their interests but simply products of regular social activities. For this, they add an additional latent variable to indicate whether a particular tweet is related or not to the user's topics of interest. As such, the latent variable adds the possibility that a particular tweet comes from another topic distribution other than the author's one.

Something similar was done by Zhao *et al.* [61] in their paper comparing Twitter and The New York Times<sup>17</sup> articles using topic models. Their proposed model for tackling Twitter posts named Twitter-LDA is an extension of LDA [4] with the assumption that every tweet relates to only one topic. As such, the generative process for a tweet can be described as; first, the user chooses a topic from its topic distribution and then chooses a Bag-of-Words, one by one, from either the topic model or an introduced background model. The background model has the distribution of background words and is common to all users. Other authors, like Sasaki *et al.* [49], have further extended this idea assuming that instead of the background distribution being the same for every user, each user has its own background distribution.

An empirical study of topic modeling in Twitter was performed by Hong *et al.* [18], comparing different aggregation strategies and topic models for Twitter data. When comparing the performance when using individual messages, aggregating them by user, and aggregating them by terms in common, the authors found that the best results came from user-based aggregation. Surprisingly, the authors also found that classifiers trained using simple TF-IDF feature vectors were significantly more accurate than feature vectors taken from the LDA model [4] or the Author-Topic model [60].

Table 3.1 (p. 19) provides a summary of the mentioned approaches for topic modeling for web and social media-based content.

Many software tools perform topic modeling of text, offering ready-to-use implementations of common topic model algorithms and word embeddings extraction, sometimes integrated with other modules for standard NLP tasks. *gensim* [62] is a widely used Python library for topic modeling, with a design goal on being fast and memory efficient. It offers various implementations of LDA-based models, word2vec feature representations and even algorithms for computing text similarity on topical features. Additionally it is possible to use it with already trained models. Another interesting tool is MALLET (MACHINE Learning for Language EXtraction) [33], a Java-based package for applications in document classification, clustering, topic modeling and information extraction. For topic model, it includes efficient routines to convert text to feature representations and implementations of algorithms such as LDA and Hierarchical LDA. It has the shortcoming of not having pre-trained models, but once trained, a model can be turned into a topic inference tool and reused on new data. Lastly, another well known tool, substantially used within the industry, is spaCy [19], a Python library for advanced natural language processing designed

---

<sup>17</sup>The New York Times - [www.nytimes.com](http://www.nytimes.com)



Table 3.1: Summary of Topic Modeling approaches for Web and Social Media-based content

Authors	Data Source	Methodology	Results
Onan [40]	Web of Science (WoS) database	Different word embeddings ( <i>e.g.</i> , word2vec, LDA2vec) and clustering ensemble.	Outperformed every model with an F1-score of 0.693.
Uteuov <i>et al.</i> [53]	VK and Facebook	Extension of the hierarchical ARTM model with NN-based word embeddings.	Overall better perplexity and coherence than the base ARTM model.
Qiang <i>et al.</i> [42]	Twitter and Google News	MRF model with word2vec and GloVe word embeddings.	Better coherence than all other models for both datasets.
Maupomé <i>et al.</i> [32]	Reddit	LDA model.	Moderate results inferior to other competition approaches.
Liu <i>et al.</i> [30]	Instagram	word2vec word embeddings and LDA model.	-
Xu <i>et al.</i> [59]	Twitter	Extension of the Author-Topic model.	Considerably lower perplexity than baseline LDA and Author-Topic models.
Zhao <i>et al.</i> [61]	Twitter and The New York Times	Extension of the LDA model named Twitter-LDA.	Better judge scores than than baseline LDA and Author-Topic models.
Sasaki <i>et al.</i> [49]	Twitter	Extension of the Twitter-LDA model with the Topic Tracking Model (TTM).	Lower perplexity than baseline LDA models and original Twitter-LDA model.
Hong <i>et al.</i> [18]	Twitter	LDA and Author-Topic model with different tweet aggregation strategies.	Best results for tweets with standard LDA and user-based aggregation.

specifically for production usage. It offers modules for a variety of NLP tasks, from simple text tokenization to more complex purposes like entity linking or morphological analysis. It contains components for text classification but also provides pre-trained models in 19 different languages, that can be used directly. Using spaCy, tmtoolkit<sup>18</sup> was design, a tool for tex mining and topic modeling with features like find/filtering topics, getting topic-word and document-topic distributions among others. A summary of these tools can be found on Table 3.2

Table 3.2: Principal Software Tools for Topic Modeling

Tool	Language	Implemented Algorithms	Pre-Trained Models	Open-Source
gensim [62]	Python	LSA, LDA, Author-Topic model and word2vec embeddings	✓	✓
MALLET [33]	Java	LDA and Hierarchical LDA	×	✓
spaCy [19]	Python	CNN text classification and word embeddings models	✓	✓

## 3.2 Forensic Technology for Human Trafficking

According to the United Nations Office on Drugs and Crime (UNODC), *Human Trafficking* is defined as “*the recruitment, transportation, transfer, harboring or receipt of people through force, fraud or deception, with the aim of exploiting them for profit*” [37]. Exploitation can take many forms, including sexual exploitation, forced labor, forced marriage, organ removal, among others.

As pointed out in their 2020 *Global Report on Trafficking in Persons* [38], with the world continuing to embrace the ever-growing digital transformation, internet technologies are used now,

<sup>18</sup>tmtoolkit - [github.com/WZBSocialScienceCenter/tmtoolkit](https://github.com/WZBSocialScienceCenter/tmtoolkit)

more than ever, for facilitating the trafficking of people. In their words, “*with the rise of new technologies, some traffickers have adapted their *modus operandi* for cyberspace by integrating technology and taking advantage of digital platforms*”. The report distinguishes the three main stages where technology is typically used and how:

- **Recruitment**, usually happens on social media and web pages. It can be passive, where victims reach out to traffickers, or active if the traffickers directly seek out the victims;
- **Advertisement**, also relying on social media and web pages, it pertains to the advertisement of job offers and exploitative services on the internet;
- **Exploitation**, typically includes the use of video equipment to stream and broadcast exploitative services.

In a positive light, the same growth in technology can be used to develop tools that help combat Human Trafficking. Most of these tools aim to aid authorities and Non-Governmental Organizations (NGOs) in their investigations, to make them more effective but also less taxing on the investigators. For example, when investigating sex trafficking, much of the police work is done through the internet, which weights on investigators that endure long hours of searching and reviewing disturbing content [23].

The following sections present the three main approaches to forensic technology in Human Trafficking. More specifically, Section 3.2.1 discusses works focused on crawling a large number of websites to extract relevant information, link, and represent it in a meaningful way. Section 3.2.2 illustrates approaches relying on the definition and labeling of data with indicators of Human Trafficking. The last approach, detailed in Section 3.2.3, directly classifies meaningful blocks of data as having signs of suspicious activity related to Human Trafficking. It is important to note that these approaches are not mutually exclusive, and solutions can be formed as a combination of them. For example, it is possible that an approach extracting and linking data from the web also uses indicators for tagging parts of the data. Similarly, an approach that classifies data containing signs of Human Trafficking can make use of data labeled prior with appropriate indicators.

### 3.2.1 Extensive Data Extraction and Linking

The most prevalent type of approach, which seems to also be the one with greater real-world adoption in investigations, aims to offer investigators a tool to more efficiently and effectively analyze large quantities of data available on the internet. This usually involves crawling websites known to be associated with Human Trafficking activity, using appropriate strategies to link the gathered data, and then building tools to query and visualize the information.

Many of the tools under this approach were developed within the DARPA MEMEX program [1], focused on the development of search technologies for better discovery, organization, and presentation of domain-specific content. The main goal was finding ways to discover relevant content in a particular domain, extending the searching capabilities to the deep and dark web, organizing and presenting the information in an immediately helpful way for investigation tasks.

Building upon earlier systems developed through DARPA, like DeepDive [36], a system architecture called DIG (Domain-specific Insight Graphs) was proposed by Szekely *et al.* [52] for building domain-specific Knowledge Graphs. Although it is built to be domain agnostic, the study uses the Human Trafficking domain as an example.

Starting with over 50 million crawled web pages, the created Knowledge Graph contains around 1.4 billion nodes. Nodes represent ads taken from escort websites and the corresponding extracted features like phone numbers, emails, ethnicities, names, and others. Edges represent either association between the ads and their features or similarities between ads given as their Jaccard Similarity.

In order to build a system that is valuable and easy to modify for various domains, it uses a page extractor module that identifies elements based on regular expressions, which can be inferred by a set of examples given by researchers. It learns by training on a collection of pages and their corresponding extractions, becoming able to extract the relevant data from similarly structured pages. For information extraction of text, a similar approach is used but complemented with the Amazon Mechanical Turk <sup>19</sup> crowdsourcing platform for generating more labeled examples.

The authors also defined an ontology to represent the terminology used for nodes and edges, being careful to have multiple representations of the same object for each possible purpose. For example, phone numbers may be represented as a string for visualizing and keyword search, but also in a structured representation for querying and analyzing. The mapping between data and the ontology was done through a previously developed technology called Karma [26], an environment to easily define data transformations for cleaning and mapping data to ontologies.

Various algorithms were used to compute the similarity between collected images and text needed for linking different ads. In regards to images, the system uses DeepSentiBank [7], an approach using deep convolutional neural networks. For text, Minhash/LSH algorithms for Jaccard Similarity allow this process to be efficient and scalable even on extensive corpora.

Lastly, the use of an ElasticSearch <sup>20</sup> NoSQL database for storing the data, allows for structured and full text search over the massive amounts of documents generated.

Kejriwal *et al.* [25] present an investigative search engine specific to the Human Trafficking domain built to be integrated into the third version of the previous DIG architecture.

The researchers had the goal of developing a system capable of answering *entity-centric* questions over noisy documents crawled from web pages of domains known to be related to Human Trafficking, like the [backpage.com](http:// backpage.com) website, later taken down by US authorities. In this case, entities can refer to victims or other latent entities like traffickers, that is, entities that are not directly present but can be inferred through patterns in the data. Therefore, *entity-centric* queries, as the name implies, revolve around entities and their attributes. An example of such a query can be knowing the number of escorts charging an hourly rate above a specific price and are also related to a given phone number.

---

<sup>19</sup>Amazon Mechanical Turk - [www.mturk.com](http://www.mturk.com)

<sup>20</sup>Elastic Search - [www.elastic.co](http://www.elastic.co)

The problem would be almost trivial if the data were stored without any noise in a relational database. However, this is not the case, and even if the data was structured in a machine-readable manner, issues like information extraction of web content still make it challenging to achieve good performance in a real-world scenario. Other problems, like the deliberate obfuscation of information by traffickers, make applying standard NLP techniques far less effective for data processing, compromising the quality of the retrieved data. As an example, retrieving phone numbers can be a complex task when written, as is sometimes the case, in a manner such as the following “(1 two 1) six 5 six - 0 9 one 2 - 21”.

The main component of the new architecture is the KGC, Knowledge Graph Construction, which takes a corpus of web documents and from it builds a Knowledge Graph, eventually with some noise, with nodes and edges that are semantically typed with classes and properties relevant to the domain.

For information extraction, the system relies on multiple NLP techniques like Conditional Random Field (CRF)-based Named Entity Recognizers to retrieve relevant attributes like hair color, eye color, and ethnicity. Regular expressions are used for phone numbers and email addresses combined with other rule-based algorithms like entity sets, dictionaries, lexicons, and external knowledge bases (*e.g.*, Geonames<sup>21</sup>).

In order to make it scalable, capable of processing millions of crawled web pages, the system takes advantage of Apache Spark<sup>22</sup> for parallelizing many operations.

The knowledge graph is indexed and stored in an ElasticSearch NoSQL database. The search engine was then designed to access the Knowledge Graph through the SPARQL Protocol and RDF Query Language (SPARQL), exposing a custom-built interface where investigators could perform queries and receive a ranked list of results based on their relevancy.

As mentioned before, the search engine was integrated into the Domain-specific Insight Graph (DIG) architecture, see Figure 3.3 (p. 23), and used by law enforcement agencies to combat Human Trafficking.

### 3.2.2 Tagging Data with Human Trafficking Indicators

Another common strategy found in the literature concerns tagging data with previously defined indicators of Human Trafficking. These indicators are meant to reflect common patterns found in Human Trafficking activity that, if appropriately assigned to segments of data, can automate part of the investigators’ work and help guide its course.

One of the projects following this technique, also supported by the DARPA MEMEX program, is presented by Kejriwal *et al.* [24] as FlagIt (Flexible and adaptive generation of Indicators for text) and end-to-end indicator mining system with the principle goal of reducing the burden and potential bias stemming from manual supervision of suspicious text, like sex ads, that can contain signs of Human Trafficking activity.

---

<sup>21</sup>Geonames - [www.geonames.org](http://www.geonames.org)

<sup>22</sup>Apache Spark - [spark.apache.org](http://spark.apache.org)

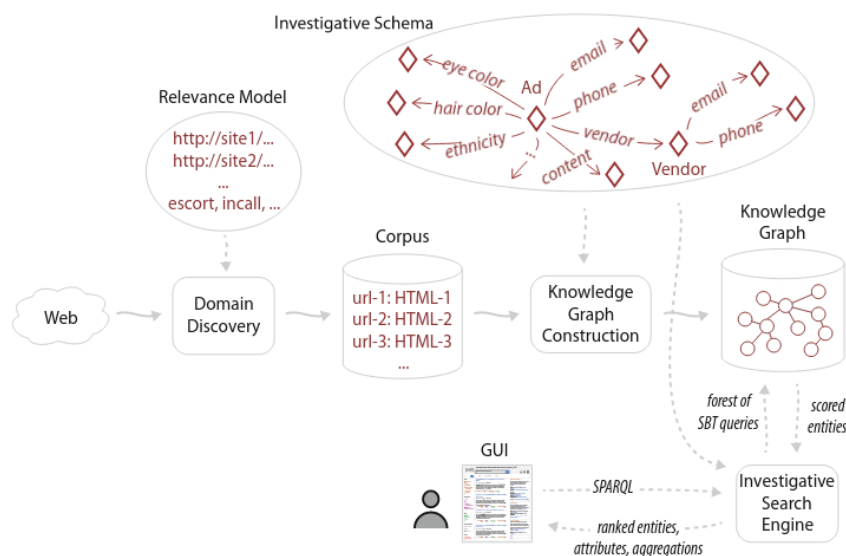


Figure 3.3: Overview of the Domain-specific Insight Graph (DIG) architecture taken from Kejriwal *et al.* [25].

FlagIt combines minimally supervised machine learning technologies, like unsupervised text embeddings and semi-supervised heuristic re-labeling.

It also begins with a web corpus taken from crawled websites related to the Human Trafficking domain. The text is scraped for descriptive content using a Readability Text Extractor (RTE) fine-tuned for high recall and further segmented into sentences. It follows with a custom Lightweight Expert System (LES) which uses shallow pattern matching rules to categorize each sentence based on how strong its predictive relationship to indicators is. These rules rely on various patterns using a Glossary, regular expressions, NLP tags, and named entities. It ends by labeling every text as either “*incall*”, “*outcall*”, “*movement*”, “*risky*” and “*multi-girl*”, using a multi-label text classification approach.

These labels are helpful descriptors for investigators that can use them both for lead generation (*i.e.*, find potential leads for new investigations) as well as lead investigation (*i.e.*, gather further information on an ongoing investigation that can serve as evidence or clues).

Another interesting tool coming from the DARPA MEMEX program was detailed by Mattmann *et al.* [31]. The research focused on taking advantage of metadata contained in images and videos to facilitate using these sorts of retrieved multimedia content to calculate similarity or perform clustering without resorting to computationally expensive and often unreliable Computer Vision techniques.

This metadata included camera and scene properties (*e.g.*, camera model serial number), color space information for video, information like bitrate and tracks. It was extracted using the Apache Tika <sup>23</sup> toolkit. Using Jaccard Similarity, these images and videos could then be compared and

<sup>23</sup>Apache Tika - [tika.apache.org](http://tika.apache.org)

clustered.

Although this work does not perform any sort of labeling of data with indicators, the novel method can help gather information crucial for detecting the presence of certain indicators. For example, if some images and videos taken from escort ads are clustered together, this could mean that they come from the same source. This is a common indicator of Human Trafficking since it can mean that the workers are being controlled by a third party, potentially a trafficker.

In their paper “*Crawling Open-Source Data for Indicators of Human Trafficking*”, Brewster *et al.* [5] propose a *Three Level* framework, see Figure 3.4, to be used as a guide for crawling the web and social media in search of indicators of Human Trafficking.

<b>Level 1</b>	<i>Credible and Accepted Trafficking Indicators</i>
	<ul style="list-style-type: none"> <li>• Forced Labor</li> <li>• Forced Prostitution</li> </ul>
<b>Level 2</b>	<i>Materials Produced as a Result of Anti-Trafficking Activities, Domain Professionals &amp; Academia</i>
	<ul style="list-style-type: none"> <li>• Cash Businesses used as fronts for prostitution and drug cultivation</li> </ul>
<b>Level 3</b>	<i>Social Media and Citizen Created Content</i>
	Observations indicating: <ul style="list-style-type: none"> <li>• Young looking employees</li> <li>• Employees appear to have been subjected to violence</li> </ul>

Figure 3.4: Three Level Framework example taken from Brewster *et al.* [5].

**Level 1** pertains to credible and accepted indicators of trafficking, based on legislation, government and non-government bodies like UNODC, EUROSTAT, and NCA. These are usually rigid, expressed using formal language, and therefore serve mainly as a guide for the other two levels, more so than as indicators that can draw new, unknown open sources of these crimes.

**Level 2** considers materials based upon the formal definitions and language of Level 1 but tailored to a broad, more diverse audience, usually found on news articles, professionals from the field, and academia. These can be used to provide weak signals or, when used in aggregation, to establish possible patterns or trends in the data.

**Level 3** is related to the terminology used in social media and web forums that originated from ordinary people. These are the best indicators to provide operational information and be used to identify specific instances of suspicious activity.

In short, Levels 1 and 2 are used more for identifying known, already reported information coming from organizations and news outlets. Level 3, on the other hand, allows for the identification of information more closely tied to perpetrators or victims that is unknown and can be used

to extract weak signals of these criminal activities.

As part of the European ePOOLICE (early Pursuit against Organized crime using enviroNmental scanning, the Law and IntelligenCE systems) project, Andrews *et al.* [2] took the framework described by Brewster *et al.* [5] and developed an Organized Crime taxonomy and combined it with entity extraction tools to filter social data that had more than a pre-defined number of signals of organized crime, and could later be manually reviewed by authorities, facilitating the access and use of social media as a new source of data for investigations.

The created taxonomy can be represented as a tree-like structure, see Figure 3.5, where each node represents a rule-set that is able to determine the relevance of the content to the subject matter. As the authors put it “*The level of specialisation of the rules themselves follows the structure of the taxonomy, moving from more generic words or phrases that may indicate Human Trafficking used at the higher levels of the taxonomy, whereas as the more specialised end of the taxonomy, more nuanced rules that may allude to the presence of criminality are used.*” [2].

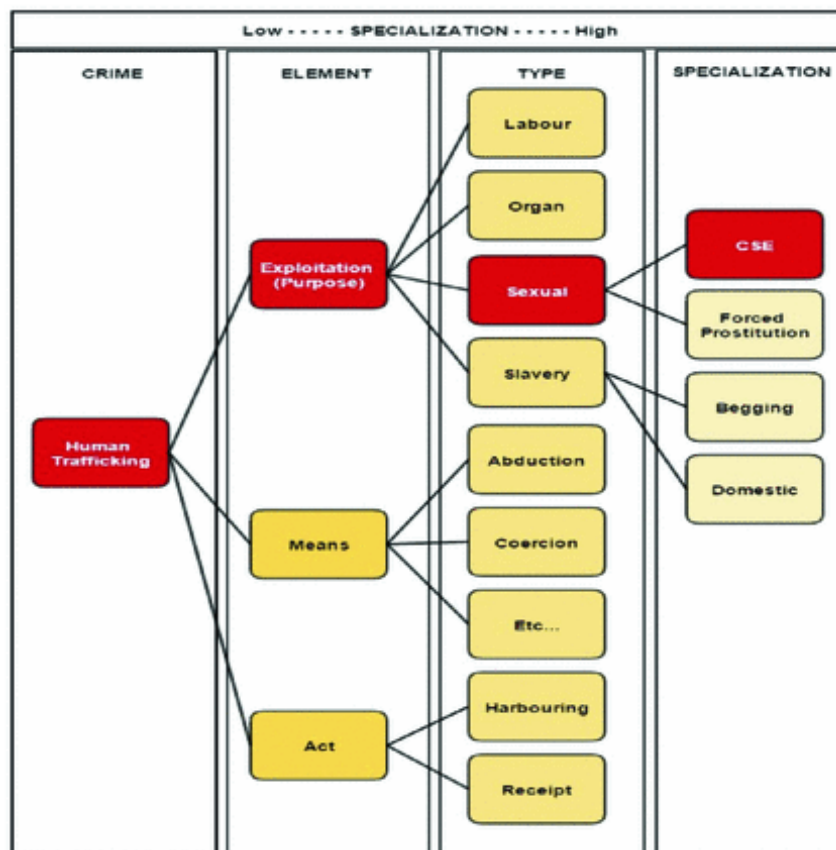


Figure 3.5: Organized Crime Taxonomy (excerpt) taken from Andrews *et al.* [2].

The approach involved using natural language processing methods for “*contextual extraction*”. Such methods are used to identify and extract key entities and facts present in the data. Afterward, a set of rules is used to try to infer relations between the facts and entities extracted.



Such rules hinge on using prepositions, POS tagging, and Boolean operators that specify the distance between words and other parameters to be able to infer relationships between the data.

Besides extracting these features from the input data, similarity techniques are used to classify content against a number of pre-defined categories. After being categorized and depending on the category, the content is subject to a rule-based filter and is ultimately discarded if it does not meet the criteria of at least one of the rules defined by the category. These rules are also “hand-crafted” and organized in a hierarchical structure, focusing on identifying keywords and phrases, with the goal of evaluating the relevance of the content in regards to the defined topics using the taxonomy structure defined.

Focusing on the United Kingdom online sex market, Giommoni *et al.* [13] attempted to identify Human Trafficking patterns in advertisements from a popular multi-service adult entertainment platform in the UK.

The authors mention two common approaches for establishing patterns of Human Trafficking, ultimately having chosen the first approach for their investigation:

- **Identifying Trafficking Indicators**, which involves detecting the presence of indicators in advertisements that render it suspicious. Indicators can include, among others, reporting different ages across multiple advertisements, reference to particular nationalities/ethnicities, only providing in-call services.
- **Automatic Collection and Categorization** usually requires using machine learning, together with data scraping, data mining, and NLP tools, to train a classifier model to discern between suspicious and non-suspicious advertisements.

For the first approach, when choosing indicators, the authors found that most studies rarely had an empirical basis, and at most, their choices appealed to authorities’ input that could not be verified. As such, knowing that it would be hard to have a solid basis to justify the chosen indicators, the authors resorted to 10 indicators, see Table 3.3 (p. 27), the first eight taken from UNODC’s list of indicators of human trafficking [38]. The last two are specific to the identification of minors, based on previous literature [20].

Ultimately, the system found that only 1.7% of all advertisements report three or more indicators. The authors point out that, in order for the indicator approach to be more effective, these indicators should be revised and standardized to be more descriptive.

### 3.2.3 Classification of Data as Suspicious Human Trafficking Activity

The last approach includes works that attempt to directly classify relevant blocks of data as containing or not suspicious signs of Human Trafficking activity. It is arguably a more difficult task since defining a threshold for what is considered enough evidence to classify as suspicious is challenging.

To assuage this ambiguity, Hernández-Álvarez [15] turned to Twitter and focused on Spanish data to identify signs of Human Trafficking of underage victims. The focus on underage victims



Table 3.3: List of Human Trafficking indicators chosen by Giommoni *et al.* [13]

	Indicator	Rationale
11	The advertisement uses third- or first-person plural pronouns	<i>“There is evidence that groups of women are under the control of others.”</i>
12	Same phone number cited in more than one advertisement	<i>“There is evidence that groups of women are under the control of others”</i>
13	High degree of similarity between sex workers’ advertisements	<i>“There is evidence that groups of women are under the control of others”</i>
14	Sex workers offer risky or violent sexual services	<i>“There is evidence that suspected victims cannot refuse unprotected and/or violent sex”</i>
15	Advertisements promoting inexpensive sex services	<i>“Receive little or no payment”</i>
16	Sex workers moving frequently across several locations	<i>“Move from one brothel to the next or work in various locations”</i>
17	Sex workers moving to different locations along with other sex workers	<i>“Live or travel in a group, sometimes with other women who do not speak the same language”</i>
18	Sex workers offering in-call services only	<i>“Show signs that their movements are being controlled”</i>
19	Advertisements using words that allude to the youthful characteristics of the sex workers	This indicator aims to uncover possible cases of underage sex workers
110	Stating a dress size that is typical of underage women	This indicator aims to uncover possible cases of underage sex workers

comes from the fact that by simply being a minor providing services of sexual nature there is already a strong indication of possible Human Trafficking. It thus makes it less questionable if it should be considered suspicious or not.

The approach began by using the Twitter API <sup>24</sup> and searching for hashtags that can be related to sexual exploitation of minors (*e.g.*, #lolita, #nueva). The collected tweets were then processed to remove non-standardized and special characters, discard tweets unrelated to the theme, removal of digits, stopwords, and correction of misspellings. Making use of syntactic analysis tools, features like recognition of third person speech, mentions of lightweight, number of adjectives and verbs were extracted alongside simpler features, including the number of hashtags, the quantity of words, and others. These were then used to classify the tweets as suspicious or non-suspicious using semi-supervised learning with Naïve Bayes and SVM algorithms.

<sup>24</sup>Twitter API - [developer.twitter.com/en/docs/twitter-api](https://developer.twitter.com/en/docs/twitter-api)



## Chapter 4

# Problem Statement and Methodology

---

4.1	Current Issues and Limitations . . . . .	29
4.2	Scope . . . . .	30
4.3	Assumptions . . . . .	31
4.4	Hypothesis . . . . .	31
4.5	Research Questions . . . . .	32
4.6	Architecture . . . . .	33
4.7	Process Flow . . . . .	33

---

On this chapter, the problem driving this dissertation is formalized and described in greater detail. It commences with Section 4.1 detailing the current issues and limitations that encouraged this project, followed by Section 4.2 that identifies the respective scope. Then, key assumptions are laid out in Section 4.3 before defining the main hypothesis in Section 4.4. Section 4.5 lists the principal research questions that guide this work. It follows with Section 4.6 that outlines the architecture for the application developed based on the proposed solution. Finally, the application process flow is detail in full in Section 4.7.

### 4.1 Current Issues and Limitations

Retrieving and analyzing information from the web is contingent on the availability and quality of the existing search tools. When it comes to data generated by people, whether on Social Media platforms or other websites, there is no concern for structuring or connecting the data outside of the approach taken by the website holding the data. Accessing this information and having it connected can be extremely useful in a plethora of situations. However, being at the mercy of the platforms storing the data, three mains issues arise:

1. **Search specificity** - There exist tools capable of accessing the information, but they are tailored to specific uses and thus do not encompass all meaningful search scenarios;

2. **Multi-platform searching** - The scenario requires searching through multiple platforms that may not even be known in order to answer a particular query adequately;
3. **Deficient search reach** - There are no tools to access the data, or the existing ones are of significantly low quality, making it difficult to make use of the information in any meaningful way.

A field where these problems are particularly limiting is Human Trafficking investigations, whether done by authorities or NGOs. This stems from the fact that valuable information can exist on the web about victims and perpetrators that could be used to find leads on existing cases or even shine a light on unknown situations. However, due to the nature of these crimes, the parties involved take special attention and care to obscure relevant information enough to make it difficult for authorities to access it readily, but not so much that potential clients cannot reach it. Moreover, platforms that benefit from having these activities conducted on them will intentionally limit the access to data in similar manners.

## 4.2 Scope

Within this dissertation, we propose an approach to facilitate the access and analysis of scattered information on the web, generated directly from people, with a particular focus on functioning as Open-Source Intelligence (OSINT) that authorities and other relevant organizations can use in the fight against crimes of sexual exploitation and Human Trafficking.

Most of the proposed solutions in the literature to extract and organize relevant information from investigative search rely on prior knowledge of platforms known to be a space where criminal activity takes place where this data is vast but accessible through tailored crawling approaches. This approach, by far the most common, relies extensively on the starting platforms. For example, in the case of Human Trafficking, many of the works begin with a list of URLs from escort and massage parlor websites, which are known to overlap with victims of sex trafficking alongside ordinary sex workers. Additionally, these websites tend to have a native search engine and filters that can be easily used to crawl through thousands of advertisements.

Deviating from this approach, this dissertation puts forth a novel approach, where the focus is on using entities found in Social Media profiles, primarily representing people or organizations, as the starting point for retrieving relevant information. This has the benefit of not requiring any prior knowledge of specific platforms where certain activities can take place. Furthermore, most of these investigations are interested in answering questions where the central figure are people, victims, or traffickers. In this approach, these entities are already at the core and it is from them where new data emerges. In other words, all data that is gathered is always linked to a particular entity (profile) from which it was found, instead of, for example, linked to a website.

### 4.3 Assumptions

Following the proposed approach to address these issues relies on some underlying assumptions, which are explained next, without which the effectiveness of this method would be put into question. These presuppositions are:

1. People who actively use the web and do so in a significant manner will tend to reveal their presence online by using Social Media platforms, more specifically, through posting links to their own content;
2. If a person is highly interested, involved or experienced in a given domain or field, they are likely to share a disproportionately high amount of content and information related to that domain, some of which could have been created by them.

Although, as far as it is known, there is no empirical evidence done to research and corroborate these assumptions, the way people seem to use Social Media suggests that they are likely to hold true. The first assumption seems reasonable since most people using the Internet are on at least one Social Media platform [8], most often several of them. Since these platforms incentivize the sharing of personal information, it is not unfounded to believe that content created on other platforms is likely to be shared through Social Media. From this, the second assumption almost logically follows. It is reasonable to believe that the more a person is involved in a particular subject, their interests will revolve around it, and subsequently, they will share a higher amount of related content.

### 4.4 Hypothesis

Taking into account these assumptions, the approach to tackle the previously mentioned problems can be formulated through the following hypothesis:

*Domain-specific information retrieval can be understood as the retrieval and processing of data relevant to answer queries pertaining a particular field. In scenarios where entities are the logical focus behind the queries, domain-specific information retrieval can be approached using Social Media as a gateway to reach relevant information by (1) identifying Social Media profiles sufficiently related to the domain, (2) collect from these profiles links to other websites associated with the entities behind the profiles, (3) extract data from both Social Media profiles as well as from the gathered websites and, (4) structure and index the data facilitating its analysis and querying.*

Using Social Media profiles as the starting point to access information is what allows the approach to not require prior knowledge of websites relevant in a domain, albeit it can still be useful. It can be easily understood why the aforementioned assumptions need to be accepted. The

effectiveness of this approach in retrieving relevant information requires people to divulge their interests and share their own created content transparently.

This approach stands out from other methods since it requires little to no prior knowledge of where the relevant information may be on the web. It can be argued that it is still helpful to know which Social Media platforms are more likely to be used by people related to specific domains. However, although true, compared to the vast amount of websites and platforms that exist, Social Media platforms are few in comparison, where the topmost used platforms already account for most activity, thus unlikely to follow a *long tail model*. As such, even though this research uses just the Twitter social media platform as a proof of concept, it could feasibly be extended to include the top Social Media platforms, extending its reach to more domains.

## 4.5 Research Questions

To prompt this research and attest the hypothesis, a set of research questions are presented that served as a guide for the development of this work, which are:

**RQ1: Can we design an algorithm capable of parsing with sufficient precision Social Media profiles that are exceptionally related to a domain by utilizing just a set of relevant domain keywords?**

Identifying relevant profiles with minimal user input allows for the tool's broader use, but doing so is not a trivial task and requires careful consideration, for example, if a high recall or high precision is preferred.

**RQ2: Can a time-efficient algorithm be devised to calculate with high precision the relatedness between a Social Media profile and an arbitrary webpage using only the contents available through them?**

People may share links to any arbitrary webpage, but for the purposes of this tool, only those that are in some meaningful way related to the profile should be considered for data extraction.

**RQ3: Can the information crawled from Social Media profiles and related web pages be provided for exploration in a usable manner?**

Once the valuable data has been identified and collected, one must consider the best approach to organize this data to facilitate the interaction between it and the user. This includes the quality and latency of the query results to ideally make it a viable tool to be used by investigators.

**RQ4: Can an architecture for the system be constructed that enables the application to be scaled while remaining performant?**

The applicability and usefulness of the tool grow with the amount of data it is capable of processing and collecting. Hence, having performance and scalability in mind when making decisions regarding its architecture and technologies will ultimately lead to a better product.

## 4.6 Architecture

The application requires the orchestration of multiple complex tasks. To guide the design, architecture and overall decision making process, the following requirements were defined and followed:

1. The application must be **modular**, making it easier to develop and test each component individually, and allowing for the possibility of future enhancements or refactors without having to rebuild the whole system;
2. The application must be **agnostic**, avoiding solutions that target solely specific domains;
3. The application must be **scalable**, remaining performant even in higher workloads.

Following these requirements, the designed architecture for the tool has a pipeline structure, where individual components or modules are responsible for specific tasks that receive inputs from other modules, perform the necessary computations and feed the results as input to other components.

The app can be first divided into the standard frontend and backend elements. The frontend or interface, written using the React framework <sup>1</sup>, is how the user interacts with the whole application, including both, triggering a search as well as visualizing its results. It communicates with the backend through a REST API, built using the FastAPI Python framework <sup>2</sup>.

The backend comprises three principal modules: the Profile Acquisition module; the Profile Selection module; and the URL Extraction and Crawling module. Per the third requirement, the last two modules are run within the Apache Spark engine <sup>3</sup>, which is able to parallelize the operations on multiple computing clusters. Additionally, all the data that is extracted is stored using the Elasticsearch engine <sup>4</sup>, which indexes the documents according to a provided schema and subsequently allows the fast retrieval of information. It also provides features such as fuzzy search and full-text search.

The overall architecture is illustrated in Figure 4.1 (p. 34). With its modular nature, the development process followed an iterative approach, where each component was built and tested separately, and only at the end were all the modules connected and the application as a whole was tested.

## 4.7 Process Flow

To better understand the application that was ultimately developed, it is essential to go through a normal flow, thus getting a broad picture of how it functions and how the various components interact. Figure 4.2 (p. 36) translates the application's process flow into a UML activity diagram and can help follow its explanation.

---

<sup>1</sup>React - [reactjs.org](https://reactjs.org)

<sup>2</sup>FastAPI - [fastapi.tiangolo.com](https://fastapi.tiangolo.com)

<sup>3</sup>Apache Spark - [spark.apache.org](https://spark.apache.org)

<sup>4</sup>Elasticsearch - [www.elastic.co](https://www.elastic.co)

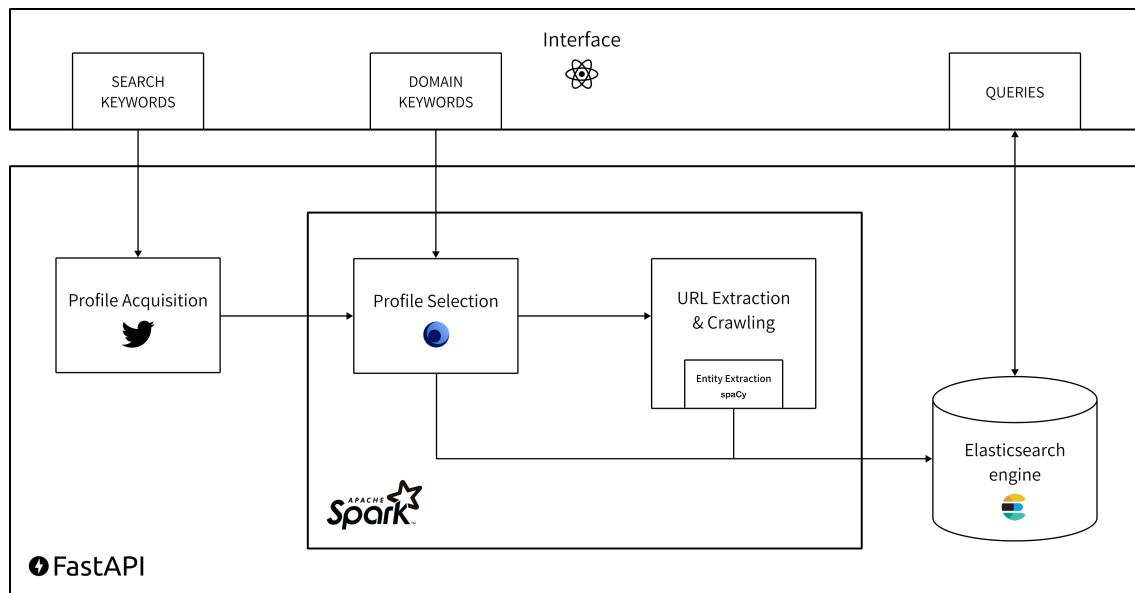


Figure 4.1: Diagram of the application's overall architecture.

As a first step, the user must use the interface to provide the search's configuration parameters. Some are mandatory, like the search and domain terms, and others are optional, like the date range to search tweets. Once at least all mandatory parameters are filled, the user can trigger a search. This will call the API's search endpoint, passing the configuration values as a JSON body. Since the application can take a while to complete, depending on the configurations, the endpoint triggers the application to run and returns the search ID. The interface will then take a pooling approach by periodically requesting the current state until the search completes and the results are available.

Once triggered, the application begins by using the Twitter API<sup>5</sup> to search for tweets containing the search terms given by the user. The search terms, although many can be used as domain keywords, are viewed separately because these can be less rigorous of the domain while still being useful. The reason is that, first, search terms can include slang or common abbreviations used in the Twitter space that would not be well known by the word embedding model used with the domain keywords. Secondly, since search terms can be combined with exclusion terms, they can include terms used by many domains and then exclude their frequently co-occurring terms in domains which are not of interest, something that cannot be done for domain terms. For each tweet returned, the respective profile ID is kept until the desired number of profiles have been collected, a limit also set by the user in the configuration.

Afterward, the Apache Spark engine is initialized to run the following modules, parallelizing their operations. It begins with the Profile Selection module, which takes in the profiles collected from the previous search and begins gathering profile data and tweets for each of them. Part of this data, namely the tweets and retweets, are viewed as documents used to extract the profile's latent topics using topic modeling. These topics, more specifically their principal terms, together

<sup>5</sup>Twitter API - [developer.twitter.com/en/products/twitter-api](https://developer.twitter.com/en/products/twitter-api)



with the profile's description, are used to parse the profiles related to the desired domain, defined by the domain terms provided by the user. This is done by calculating their cosine similarities to the topic terms. If the resulting score is above a given threshold, the profile is kept for the next module, together with the collected data. Otherwise, it is discarded.

Still within the Apache Spark engine, the URL Extraction & Crawling module will take the selected profiles and begin crawling the links found present in their tweets. While crawling, it collects the corpus of each website together with some metadata it can find in HTML tags. After crawling each link by a given depth, it uses regular expressions with fuzzy matching to look for the user's username and Twitter name in the corpus. Each website is scored accordingly depending on the number and type of matches. Similar to the previous module, if the score surpasses a given threshold, the website is understood to be related to the profile, and its information is kept. Furthermore, depending on the entity types the user selected in the configuration, named entity extraction is performed on the related websites' corpus to provide even more valuable and structured data. In the event that the website is a "*link-tree*" website, that is, a website whose purpose is to aggregate links to different platforms where the person is present, these links are collected and added to also be crawled.

Since requests to the Twitter API are limited, and crawling websites can be an expensive task, the data extraction occurs mainly in tandem with the previous modules, instead of waiting to have the selected profiles and related links to begin extraction. This includes, for example, the entities detected by Twitter on each tweet or images found on the crawled websites. By the end of the previous module, each profile already has associated a document containing the extracted data from the multiple phases. This document is compliant with a schema defined in an index of the Elasticsearch engine, allowing for it to be sent to the engine ready to be indexed appropriately.

At this point, the search is completed, which is reflected in the UI. The user can now analyze the results that were gathered and query the documents to look for the presence of particular terms or entities (*e.g.*, emails, phone numbers, or links).

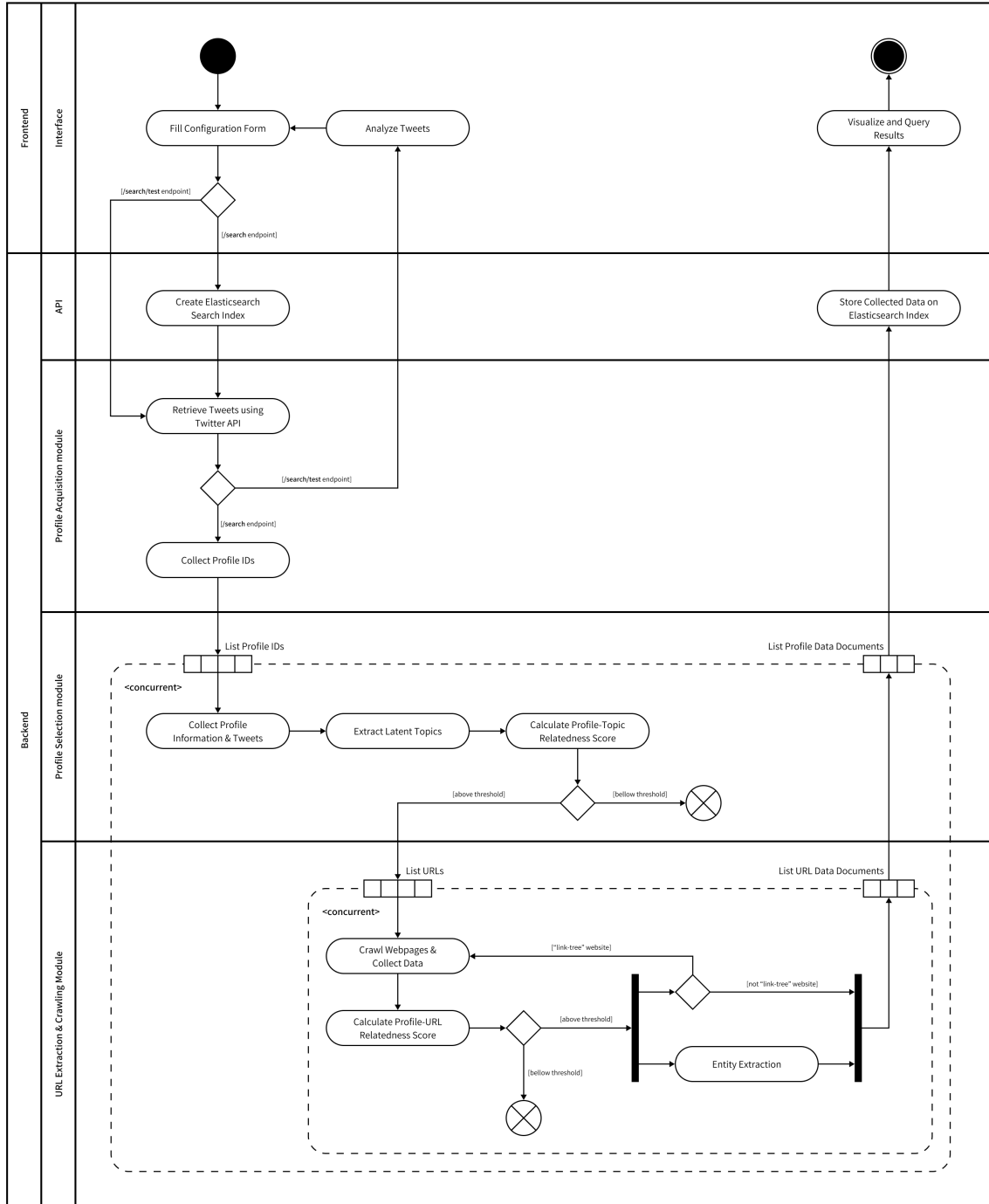


Figure 4.2: UML activity diagram of the application's process flow.

## Chapter 5

# Implementation

---

5.1	Profile Acquisition module . . . . .	37
5.2	Profile Selection module . . . . .	39
5.3	URL Extraction & Crawling module . . . . .	53
5.4	Data Storage and API . . . . .	66
5.5	Interface . . . . .	73

---

In the previous chapter, the problem that underpins this work was detailed, and the proposed solution was put forth, as well as the architecture and flow of the resulting application. This chapter describes the different modules that make up the application in greater detail. It begins with the three principal modules, starting with the Profile Acquisition module in Section 5.1, followed by the Profile Selection module in Section 5.2 which also includes the detailing of the work necessary to perform its evaluation and the corresponding results. Similarly, the URL Extraction & Crawling module's algorithm and its evaluation are described in Section 5.3. It follows with Section 5.4 that provides a thorough outline of the application's data storage mechanism and the API that powers the communication to the backend. Lastly, Section 5.5 showcases the features and various views from the developed interface.

### 5.1 Profile Acquisition module

The Profile Acquisition module is a simple module responsible for fetching an initial batch of profiles that can be of interest. In order to enact this, it requires the user to have provided the following set of parameters:

- **profiles** - the number of profiles to collect, although, depending on the rest of the configuration, it is possible that this number cannot be reached;
- **keywords** - an array of terms to search for in tweets;

- **hashtags** - an array of terms similar to *keywords*, but they must appear in tweets as hashtags (*i.e.*, preceded by a “#”);
- **exclude** (*optional*) - an array of terms that cannot appear in the returned tweets;
- **countries** (*optional*) - an array of country’s ISO 3166-1 alpha-2 codes, which will filter the returned tweets by just including those tweeted within these countries;
- **languages** (*optional*) - an array of IETF BCP 47 language tags, which will filter the returned tweets by those written in these languages;
- **start\_time** (*optional*) - a string representing the earliest date, in ISO 8601 format (*i.e.*, “YYYY-MM-DDTHH:mm:ssZ”), when the returned tweets could have been posted;
- **end\_time** (*optional*) - a string representing the latest date, in ISO 8601 format (*i.e.*, “YYYY-MM-DDTHH:mm:ssZ”), when the returned tweets could have been posted;

These configurations are first converted into a query that is compliant with the Twitter’s query guidelines<sup>1</sup>. So, for example, the following configuration:

```
"searching": {
  "profiles": 1000,
  "keywords": ["tennis match", "grand slam", "tennis tournament"],
  "hashtags": ["ausopen", "usopen", "wimbledon"],
  "exclude": ["golf", "table", "badminton"],
  "countries": ["US", "CA"],
  "languages": ["en"],
  "start_time": "2022-05-01T00:00:00Z",
  "end_time": "2022-05-20T00:00:00Z"
}
```

Listing 1: Example of a possible configuration for the searching portion.

is converted to the following query string:

```
((tennis match OR grand slam OR tennis tournament) OR
(#ausopen OR #usopen OR #wimbledon)) -golf -table -badminton
(place\_country:US OR place\_country:CA) (lang:en)
-is:retweet -is:reply -is:nullcast
```

Listing 2: Example of query string for searching tweets using the Twitter API.

It automatically adds the necessary string to prevent the tweets from being retweets, “-is:retweet”, replies, “-is:reply”, or tweets created for promotional ads, “-is:nullcast”.

Since the starting and end times are optional fields, if the user does not define them, they default to thirty days and thirty seconds prior to the current time, respectively.

<sup>1</sup>Twitter Query guidelines - [developer.twitter.com/en/docs/twitter-api/tweets/counts/integrate/build-a-query](https://developer.twitter.com/en/docs/twitter-api/tweets/counts/integrate/build-a-query)

This query is passed to the Twitter API's tweet search endpoint to begin retrieving tweets. For each tweet, the corresponding user is added to a set of user IDs. This set is maintained, discarding duplicates, until the requested number of users is reached or an API request limit, defined based on the number of users, is surpassed. At this stage, the only filtering is to discard profiles with less than 500 total tweets. This is necessary because the application requires a minimal number of tweets to be gathered in order to achieve some accuracy in extracting latent topics. Furthermore, profiles with very few tweets are unlikely to be active and, therefore, applicable since the goal is to gather and analyze the links shared by the user.

## 5.2 Profile Selection module

The Profile Selection module is a more complex module that takes the initial batch of profile IDs gathered in the previous module and identifies the ones related to the wanted domain. At the same time, it extracts relevant information from the profile, such as named entities found in the tweets, in the event that it is selected.

The module's flow is described in Section 5.2.1, explaining the module's algorithm in detail.

The development of this model followed an experimental approach. That is, decisions like which models to use, heuristics for the algorithm, and threshold values for filtering profiles were defined by trying different configurations and analyzing the module's performance in classifying the profiles.

However, in order to correctly evaluate each configuration, a dataset was first created. Subsection 5.2.2 documents the steps involved in the creation of said dataset and subsequently details how it was used to evaluate different configurations of the algorithm and the reasoning behind the choices made.

### 5.2.1 Process Flow

As previously described in Subsection 4.7 of Chapter 4, the Profile Selection module is responsible for discerning social media profiles related to a given topic, using only the publicly available information present on the profile, such as posts or replies, and a user given set of keywords. It can be seen as a binary classifier since, for a set of keywords, it will classify profiles as either related or not related to the underlying topic. Since this project only uses the Twitter social media platform, this module is targeted at Twitter profiles. Nevertheless, the underlying ideas and indeed all stages can be adapted to other social media applications.

In order for this module to work, the user must provide beforehand the following two parameters, both mandatory:

- **keywords** - an array of terms that should closely relate to and distinguish the domain of interest;
- **tweets\_per\_profile** - the maximum number of tweets to collect from each profile that must fall between a minimum of 500 and a maximum of 3,200 (a limit imposed by Twitter);

The process with which the Profile Selection module classifies profiles can be subdivided into four stages: (1) Extraction; (2) Processing; (3) Topic Discovery; (4) Classification.

The **Extraction phase**, as the name suggests, encompasses the extraction of the necessary information from a Twitter profile using the Twitter API. Some of this information is necessary for classifying the profile. Namely, it will gather the profile’s description, if one exists, and will request up to the maximum number of tweets set in the configuration by the user. The profile’s description and collected tweets will be used later to filter the profiles related to the domain.

In addition, it also extracts information about the profile that will be useful for the next module, URL Extraction & Crawling. More specifically, it stores the username, name, and all links found in the processed tweets. The Twitter API allows for requesting metadata about each tweet, one of which is the set of URLs detected. This set is requested for each tweet, however, not all links are accepted. First, every URL is “normalized”, that is, white spaces are stripped, and if the last character is a slash, it is removed. This normalization helps the accurate detection of duplicates since any character variation, like an extra slash or white space, can make two identical URLs appear different. The normalized URL is then checked if it is valid using the Python’s validators package<sup>2</sup> and if it has been seen before, done through a bloom filter. Finally, the URL is rejected if its hostname is “**twitter.com**”. This is because Twitter considers the URL of images on a tweet to be a part of that tweet’s set of links. This resulted in many of the extracted URLs being images hosted on Twitter that were not useful for later crawling.

Besides the later information collected from a profile, other metadata is also stored. Albeit this data is not used in any module for a specific purpose, it is still gathered since it may be helpful for the end-user, depending on the use case. This information includes the profile image URL, the location, and named entities detected in the tweets. Like the set of URLs, these entities can also be requested for each tweet and have been previously identified by Twitter, requiring no extra processing besides removing duplicates.

In this phase, tweets detected as being written in a language other than English are discarded. Additionally, they are grouped into tweets and retweets. Retweets are usually written by other profiles but can still be valuable to identify the user’s topics, given that users tend to also retweet information related to their topics of interest.

The **Processing phase**, which happens in tandem with the Extraction phase, takes the text in the extracted tweets and retweets and processes it to make it more suitable for the later topic extraction and classification phases. More specifically, it performs the following, in order, for each string of text:

1. Removes all URLs using a regular expression;
2. Removes sanitized characters that Twitter uses to replace certain characters (*e.g.*, Twitter converts “&” to “&amp;”);
3. Removes the retweet tag, if present, which is the string “RT” at the beginning, used to signal that the tweet is, in fact, a retweet;

---

<sup>2</sup>validators - [github.com/kvesteri/validators](https://github.com/kvesteri/validators)

4. Removes all mentions, which are Twitter handles of other profiles prefixed with an “@” character;
5. Converts every character to lowercase;
6. Removes all HTML tags;
7. Removes all punctuation characters;
8. Removes all numerical and non-alphanumeric characters;
9. Replaces multiple white space characters with a single one;
10. Removes stopwords using Gensim’s default stopwords list <sup>3</sup>;
11. Removes all words that are less than three characters long since one and two-character words are common but unlikely to help identify topics;

This stage is crucial to distill the text as much as possible to only the most relevant words, resulting in more meaningful and accessible topic extraction.

In the **Topic Discovery** phase, the text extracted from the tweets and retweets is tokenized and lemmatization is performed using the WordNet lemmatizer <sup>4</sup>. As explained in Section 3.1 (p. 12) of Chapter 3, topic models work on a collection of documents. Furthermore, since tweets have a character limit, currently 280 characters, these tend to be short, especially once common words are removed. A widely used approach to achieve better results is to aggregate multiple tweets and view them as documents. In this case, tweets are aggregated in groups of five in the order they were collected.

The tokens are then filtered so that words that appear in less than 10% or more than 99% of profile’s documents are discarded. The choice to only remove the top 1% of words stems once more from the short nature of tweets. Users already tend to summarize their words and use the most necessary ones given the character limit. Since the goal is to discover the most prevalent topics, discarding many top words incurs the risk of removing crucial topical words.

Finally, before applying the model, each document is converted into its Bag of Words representation, (cf. Section 2.1.2, p. 6). The corpus is then fed into Gensim’s Ensemble LDA model <sup>5</sup>. Since LDA is non-deterministic, the topics identified may vary with each run. In an attempt to single out only the most stable topics, the Ensemble LDA approach effectively trains multiple LDA models and considers only those that emerge multiple times. This makes it less likely that a topic distribution results from the model’s natural variance. If stable topics are found, these are returned as an array of terms and their corresponding probabilities. The ten most important terms from each of the top-5 topics are saved, that is, the ten terms identified with the highest probability of appearing. Topic extraction is done separately for tweets and retweets, each having its corpus.

<sup>3</sup>Gensim stopwords - [radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.remove\\_stopwords](http://radimrehurek.com/gensim/parsing/preprocessing.html#gensim.parsing.preprocessing.remove_stopwords)

<sup>4</sup>WordNet lemmatizer - [www.nltk.org/\\_modules/nltk/stem/wordnet.html](http://www.nltk.org/_modules/nltk/stem/wordnet.html)

<sup>5</sup>Ensemble LDA - [radimrehurek.com/gensim/models/ensemblelda.html](http://radimrehurek.com/gensim/models/ensemblelda.html)

Finally, the **Classification** phase, in order to score the profile, needs the set of keywords provided by the user, which are meant to accurately represent the desired domain, the set of terms for each topic collected prior, and the profile's description.

The profile's description, if it exists, goes through the same text processing as each tweet as it is collected. However, it is not lemmatized since it is not used for topic extraction as it is usually just one or two sentences, not nearly enough to perform topic modeling. Albeit this, the description is still widely used as a means to do a short introduction to the profile and the person behind it. We can therefore assume that it will often contain terms closely related to the profile's interests (*e.g.*, a journalist is likely to include their profession and main area of expertise in their description). Because of this, after its processing, the description is interpreted as an array of terms that also represent a profile's topic. Effectively, it functions like one of the sets of topical words extracted in the previous phase. These sets of terms must first be converted to a vector representation using a word embedding model, (*cf.* Section 3.1.2, p. 15).

The user-provided keywords can contain more than one word. For example, if the domain is tennis, one of the provided keywords could be "*tennis tournament*". This keyword has to be interpreted and used as the word pair. If used individually, the word "*tournament*", although still related to the desired domain, has a broader context. It is a term used in multiple sports and could therefore influence the classifier to accept profiles related to sports in general and not just tennis. This is where one of the essential characteristics of semantically meaningful word embedding models is used. Namely, the fact that vector operations, such as addition and subtraction, in the word embedding space, preserve the semantic meaning of the terms they manipulate. In this example, the vector representations of the words "*tennis*" and "*tournament*" are added. Conceptually, this should mean that the position of the vector for "*tournament*", presumably in some high dimensional space related to sports in general, when added to the vector for "*tennis*", results in a position still near terms widely used in tournaments, but now much closer to terms specifically used in tennis tournaments.

To better exemplify this, Listing 3 (p. 43) demonstrates the results of vector operations on the distances (or similarities) between terms. It first calculates the vector representations of the words "*tournament*", "*tennis*", "*racket*", "*chess*" and "*pawn*". Then, two new vectors are calculated by summing "*tournament*" with "*tennis*" and "*chess*". As expected, the cosine similarity between "*tennis tournament*" and "*racket*" is significantly larger than just between "*tournament*". The same happens with "*chess tournament*" and "*pawn*". Moreover, "*tennis tournament*" is not that much closer to "*pawn*" than "*tournament*", nor is "*chess tournament*" to "*racket*".

The important conclusion here is that by adding up the vector representations of composite keywords, the semantic properties are preserved, and the resulting vector can be used as if it was a single term.

Finally, once all terms are converted, the algorithm iterates through each keyword vector and calculates the cosine similarities between it and the vectors from each topic's terms. The four best scoring similarities for each topic are averaged, only keeping the highest topic average for each keyword. Once every keyword has its corresponding score, the bottom 30% scores are discarded,



```

>>> tournament = word_model.get_vector("tournament")
>>> tennis = word_model.get_vector("tennis"),
>>> racket = word_model.get_vector("racket")
>>> chess = word_model.get_vector("chess"),
>>> pawn = word_model.get_vector("pawn")

>>> tennis_tournament = tennis + tournament
>>> chess_tournament = chess + tournament

>>> print(model.cosine_similarities(tournament, [pawn, racket]))
[0.0603085, 0.19315435]
>>> print(model.cosine_similarities(tennis_tournament, [pawn, racket]))
[0.09073523, 0.34008768]
>>> print(model.cosine_similarities(chess_tournament, [pawn, racket]))
[0.34936872, 0.254306]

```

Listing 3: Example of how vector operations in word embedding space meaningfully influence distances.

and the remaining ones are averaged. This process is done for the description, tweets' topics, and retweets' topics, each having its score. The profile's overall score is simply the average of these three results. If some are impossible to calculate, for example, the profile does not have a description, or the Ensemble LDA model could not find stable topics for retweets, the average of the remaining available scores is taken. The final score is always a number between 0 and 1.

To better describe the calculation of the overall score, let us imagine a profile that just has topics from its tweets, so no description or retweet's topics. In this scenario, the overall score is just the score from the tweets' topics. From the tweets, the model found two topics  $T_A$  and  $T_B$ , each having ten topical terms, so  $t_{1A}$ ,  $t_{2A}$  until  $t_{10A}$ , and the same for  $T_B$ . Additionally, the user provided two domain keywords  $t_{1D}$  and  $t_{2D}$  (in an actual search this number should be higher). First, each term is converted to its vector representation using the word embedding model, so  $t_{1A}$  would become the vector  $v_{1A}$ ,  $t_{1D}$  to  $v_{1D}$  and so on. Then, the cosine similarity between each domain keyword vector and the topical terms' vectors is calculated, keeping only the four best similarities for each topic and then averaging them. So, for  $v_{1D}$ , the four highest similarity scores for topic  $T_A$  could be with the terms  $t_{2A}$ ,  $t_{4A}$ ,  $t_{6A}$  and  $t_{8A}$ , resulting in the average  $avg1_{DA}$ . The same can then be done for topic B, resulting in the average  $avg1_{DB}$ . Of these two averages, only the best one is kept for each domain keyword, so if  $avg1_{DA}$  is larger than  $avg1_{DB}$ , only it is kept. For the second domain keyword, it could be that  $avg2_{DA}$  is smaller than  $avg2_{DB}$ , thus only the average for the second topic is kept. In the end, for each domain keyword, there should be a corresponding score, in this scenario,  $avg1_{DA}$  and  $avg2_{DB}$ . Since there are only two domain keywords, the final score is the average of both domain keywords' scores. If, however, there were ten domain keywords, the worst three scores would be discarded, and the final score would be the average of the remaining seven.

To determine if a profile is related to the desired topic, its overall score must be larger or equal

to a pre-defined threshold value. If it is, the profile and its corresponding information are kept. Otherwise, it is discarded.

In Figure 5.1, a diagram illustrating the scoring algorithm behind the Classification phase helps to visualize how the different components interact to calculate the final score from the initial profile information.

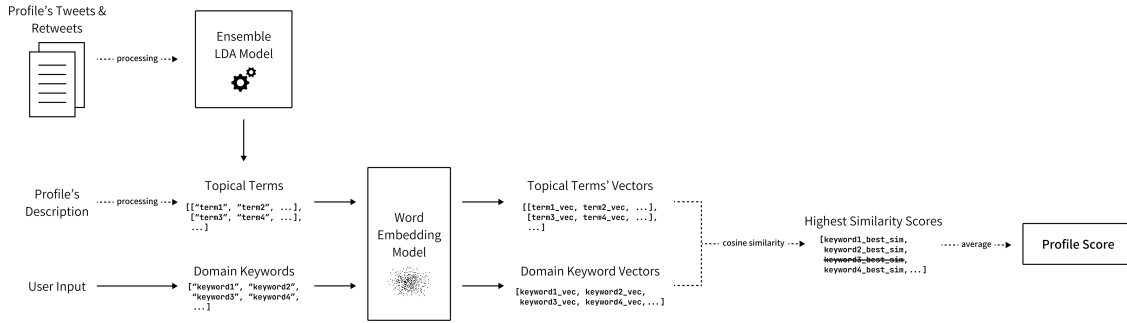


Figure 5.1: Diagram of the Profile Selection module's Classification phase process flow.

## 5.2.2 Evaluation

This module was evaluated using a custom built dataset, detailed in Subsection 5.2.2.1. It served to validate the choice of the word embedding model, see Subsection 5.2.2.2, the topic sources used, see Subsection 5.2.2.3, and the heuristics applied in the classification, see Subsection 5.2.2.4. Lastly, it was used to empirically set a threshold value for the classifier, explained in Subsection 5.2.2.5.

### 5.2.2.1 Dataset

The module was tested using six different topics, “*Ambient Music*”, “*Climate Activism*”, “*Quantum Information*”, “*Contemporary Art*”, “*Tennis*”, and “*Information Retrieval*”. Before testing the effectiveness of the classifier, a dataset had to be created of these topics, and Twitter profiles, more specifically Twitter handles, that are closely related to each topic.

The approach began by finding lists of people well known in the chosen fields. Table 5.1 (p. 45) showcases the fields chosen and the corresponding sources where the list of people was gathered.

Once a list of names was gathered, the following step was to discover the Twitter profiles of the corresponding people. For this, the Twitter API was used to search for users using the names and retrieve the first profile returned. Each profile was then manually analyzed to confirm that it actually corresponded to people related to the topic in question. Otherwise, it was rejected.

In the end, a total of 271 profiles were gathered from over a thousand names. Table 5.2 (p. 45) breaks down the number of profiles in each field.

Table 5.1: Sources for list of people related to specific fields.

Domain/Topic	Source
Ambient Music	<a href="https://en.wikipedia.org/wiki/List_of_ambient_music_artists">en.wikipedia.org/wiki/List_of_ambient_music_artists</a>
Climate Activism	<a href="https://en.wikipedia.org/wiki/List_of_climate_activists">en.wikipedia.org/wiki/List_of_climate_activists</a>
Quantum Information	<a href="https://en.wikipedia.org/wiki/Category:Quantum_information_scientists">en.wikipedia.org/wiki/Category:Quantum_information_scientists</a>
Contemporary Art	<a href="https://en.wikipedia.org/wiki/List_of_contemporary_artists">en.wikipedia.org/wiki/List_of_contemporary_artists</a>
Tennis	<a href="http://www.atptour.com/en/rankings/singles">www.atptour.com/en/rankings/singles</a> ; <a href="http://www.wtatennis.com/rankings/singles">www.wtatennis.com/rankings/singles</a>
Information Retrieval	<a href="http://sigir.org/awards/sigir-academy">sigir.org/awards/sigir-academy</a>

Table 5.2: Distribution of the gathered profiles by topic.

Field/Topic	Number of profiles
Ambient Music	102
Climate Activism	69
Quantum Information	9
Contemporary Art	26
Tennis	52
Information Retrieval	13
<b>Total</b>	<b>271</b>

After gathering this list of topics associated with Twitter profiles, it was also necessary to associate each topic with a set of keywords that accurately represent it. The set of keywords used for all subsequent tests was defined using each topic's Wikipedia page and manually selecting relevant words. It can be viewed in Table 5.3 (p. 46).

With a dataset now available, different algorithm configurations were tested to analyze and improve its efficacy. Among multiple parameters that could be changed, the two most relevant decisions were (1) which word embedding model to choose for calculating the similarities between words and (2) what threshold value to set for distinguishing between related and unrelated profiles.

In order to compare different configurations of the classifier, the Receiver Operating Characteristic (ROC) curve was used, combined with the Area Under the Curve (AUC) value. The ROC curve is a widely used graph for visualizing the diagnostic ability of a binary classifier. It plots the True Positive Rate (TPR) in relation to the False Positive Rate (FPR) for multiple threshold values between a given range. In combination with the ROC curve, a numerical way to measure the predictive power of the classifier is to calculate the area under the ROC curve. A random classifier will have an AUC value of 0.5, whereas a perfect classifier will have an AUC value of 1.

Table 5.3: Set of keywords used for each topic.

Field/Topic	Keywords
Ambient Music	“ambient music”; “ambient techno”; “ambient house”; “biomusic”; “chill-out”; “atmospheric”; “downtempo”; “nature soundscapes”; “electronic”
Climate Activism	“climate change”; “climate activism”; “climate movement”; “environmental movement”; “green future”; “climate justice”; “protesting”; “save planet”; “environment”; “paris agreement”
Quantum Information	“quantum information”; “quantum system”; “quantum theory”; “computer science”; “quantum mechanics”; “cryptography”; “quantum information processing”; “qubits”; “quantum computing”
Contemporary Art	“contemporary art”; “mixed medium”; “painting”; “modernism”; “impressionism”; “art gallery”; “art museum”; “art piece”; “avant-garde”; “art collector”; “painting”; “abstract painting”; “minimalism”; “new media”
Tennis	“tennis player”; “tennis”; “racket”; “sport”; “tennis court”; “grand slam”; “tennis tournament”; “ATP”; “australian open”; “us open”; “rolland garros”; “wimbledon”
Information Retrieval	“information retrieval”; “information system”; “information science”; “computing”; “document searching”; “metadata”; “web search”; “data extraction”

The closer the AUC value is to 1, the better the classifier’s supposed diagnostic ability.

For each test, the similarities were calculated first against the correct topic and then against one of the other remaining topics. Any given profile was tested against the same wrong topic in all tests. The distribution of wrong topics was uniformly distributed. That is, for any topic, the number of its profiles tested against another wrong topic was around the same for any of the other topics. Following it, the TPR and FPR were calculated for all threshold values between 0 and 1 with an interval step of 0.025.

### 5.2.2.2 Word Embedding Model

Gensim maintains a repository <sup>6</sup> of datasets and pre-trained word embedding models that can be used directly.

Not all available word models are useful for this application. Some, for example, are trained to be used for concept extraction of phrases. Of the ones that can be used, many are variations of a single model, trained with different amounts of data, resulting in different embedding sizes. Out of all available models, the following four were tested:

- **FastText model** trained with text taken from Wikipedia 2017, the UMBC WebBase corpus and statmt.org news dataset;
- **GloVe model** trained with 2,000 million tweets;

<sup>6</sup>Gensim-data - [github.com/RaRe-Technologies/gensim-data](https://github.com/RaRe-Technologies/gensim-data)

- **GloVe model** trained with a corpus from Wikipedia 2014 and Gigaword fifth edition;
- **word2vec model** trained with text taken from Google News.

Table 5.4 summarizes a few relevant characteristics of these models.

Table 5.4: Characteristics of tested word embedding models.

Model	Embedding size	Number of vectors	Size	Number of tokens	Text source
fasttext-wiki-news-subwords	300	999,999	958 MB	16,000 million	Wikipedia 2017; UMBC Web-Base corpus; statmt.org news dataset
glove-twitter	200	1,193,514	758 MB	27,000 million	Twitter
glove-wiki-gigaword	300	400,000	376 MB	6,000 million	Wikipedia 2014; Gigaword 5
word2vec-google-news	300	3,000,000	1,662 MB	100,000 million	Google News

The TPR and FPR are calculated for all four models, using the average of the similarities obtained for the description, tweets' and retweets' topical terms.

Figure 5.2 shows the ROC curves obtained for all 4 models. When comparing the models, it is evident that the best performance comes with using the word2vec model. Conversely, of the other three models, the GloVe and FastText models trained on Wikipedia appear to have similar diagnostic power, with the GloVe model trained on tweets having fairly worst results.

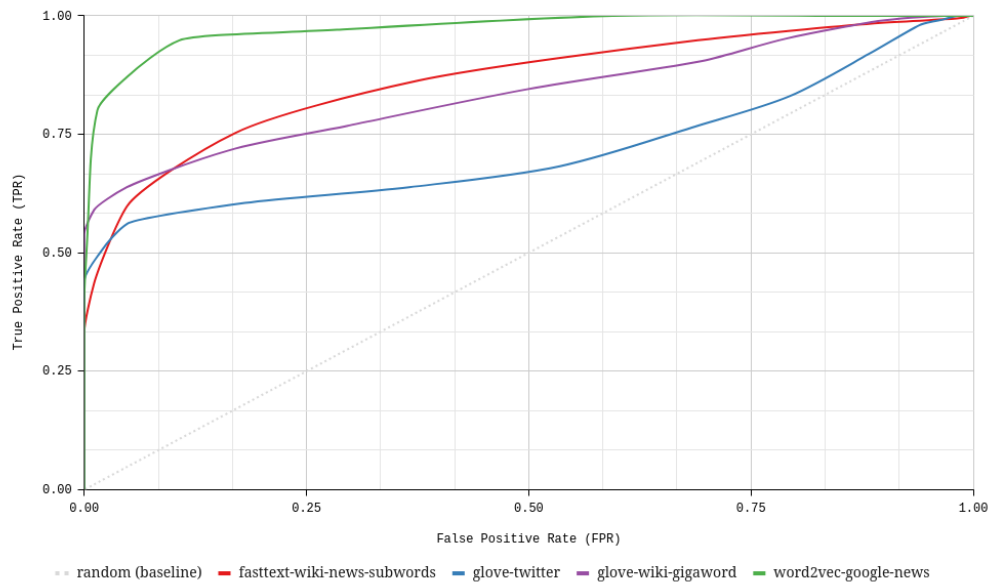


Figure 5.2: ROC curves obtained from the different word embedding models.

To more rigorously demonstrate the differences in performance between the models, Table 5.5 (p. 48) presents the AUC values obtained with each ROC curve.

Table 5.5: Area Under the Curve (AUC) values for the tested word embedding models.

Model	Area Under the Curve (AUC)
fasttext-wiki-news-subwords	0.901
glove-twitter	0.743
glove-wiki-gigaword	0.863
<b>word2vec-google-news</b>	<b>0.983</b>

Given these results, the word2vec model clearly outperforms the other models and therefore was chosen as the word embedding model for the algorithm. Multiple factors could explain word2vec’s superiority. Looking at the characteristics of each model, (*cf.* Table 5.4, p. 47), word2vec was trained with the highest number of tokens out of all, nearly four times as many tokens as the second highest. As a result, which is evidenced by the fact that it also has the highest number of vectors, its vocabulary is the most extensive. Together with its substantial embedding size of 300, it can capture relationships and, therefore, similarities between words more accurately, besides having more breath of “knowledge”.

For all the ensuing tests in the remaining sections, the word embedding model used was the Google News word2vec model.

### 5.2.2.3 Topic Sources

As mentioned previously, in the final algorithm, a profile’s score is an average of the scores obtained by the similarities between the keywords and the description, tweets’ and retweets’ topics words. It is important to restate that no topic extraction is done for the description, as it is understood to be the term distribution of a topic already.

Before choosing to use all three similarities for the final score, the classifying ability of each one had to be accessed to determine if all could and should be used to score a profile.

Figure 5.3 (p. 49) shows the plotting of the ROC curves for all three components and their average. It confirms that the average of all three scores does offer better results than any of them individually.

From analyzing the graph, one can verify that the ROC curve of the average of the three components is better than any of them on their own. Interestingly, the description seems to have the least classifying power, which makes sense. It is the component most likely to have just a few words, and many profiles have a description that does not reflect their primary interest. Some have jokes or quotes, while others have links to pages that better reflect their work. Nevertheless, it is common to find descriptions containing meaningful topical words that result in a very accurate similarity score. Another remarkable conclusion is that the text taken from a profile’s retweets seems to have more diagnostic capability than the profile’s own tweets. A possible explanation for

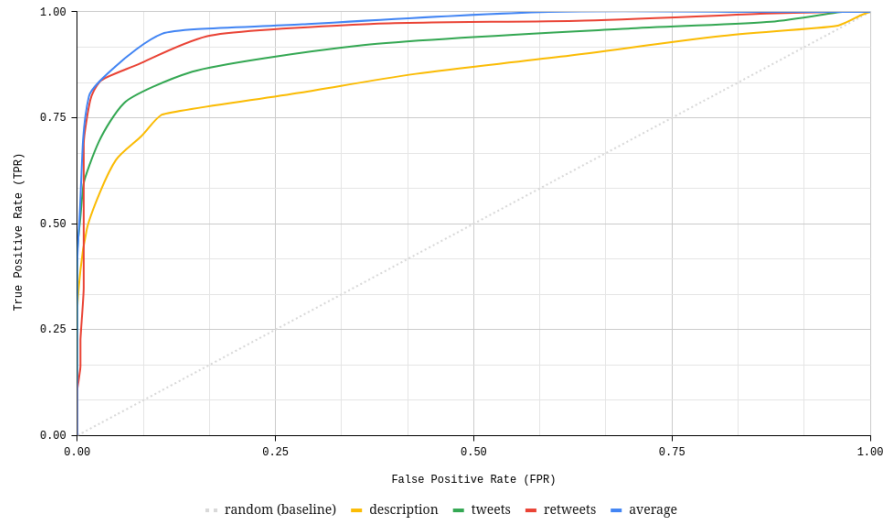


Figure 5.3: ROC curves obtained, using the word2vec model, for the profiles' description, tweets and retweets similarities.

this is that, although people occasionally tweet random thoughts or events in their lives, retweets may happen more so when the person finds another tweet interesting or relevant to them. As such, it could be the case that the retweets better reflect the profile's interests and are generally more descriptive and longer. A more straightforward explanation can be that retweeting is easier than tweeting, which leads to more text being extracted from the former, thus having a more extensive corpus which improves the topic extraction.

The AUC values shown on Table 5.6 confirm the perceived quality of the different classifiers.

Table 5.6: Area Under the Curve (AUC) values for the different profile components.

Component	description	tweets	retweets	average
AUC	0.873	0.937	0.967	<b>0.983</b>

#### 5.2.2.4 Heuristics

The previous description of the algorithm explains the filtering done while calculating the similarity scores for a given component, encompassing a couple of heuristics. Namely, after calculating the cosine similarity between a given keyword and the topic's terms, only the top four scores are considered and averaged. Furthermore, once this score is obtained for all given keywords, the 30% worst scores are discarded before returning the remaining average, which constitutes that component's score. It is essential to confirm that both these heuristics positively affect the classifier's quality.

As before, to analyze the effects of these heuristics, the ROC curves of the classifier were plotted for the different scenarios. More specifically, when no heuristics are applied, “*none*”; when only the top four scoring words are considered, “*top-4 topic words*”; when the bottom 30% of keywords are ignored, “*discard 30% worst keywords*”; and when both heuristics are applied, “*both*”. The corresponding graph can be seen in Figure 5.4.

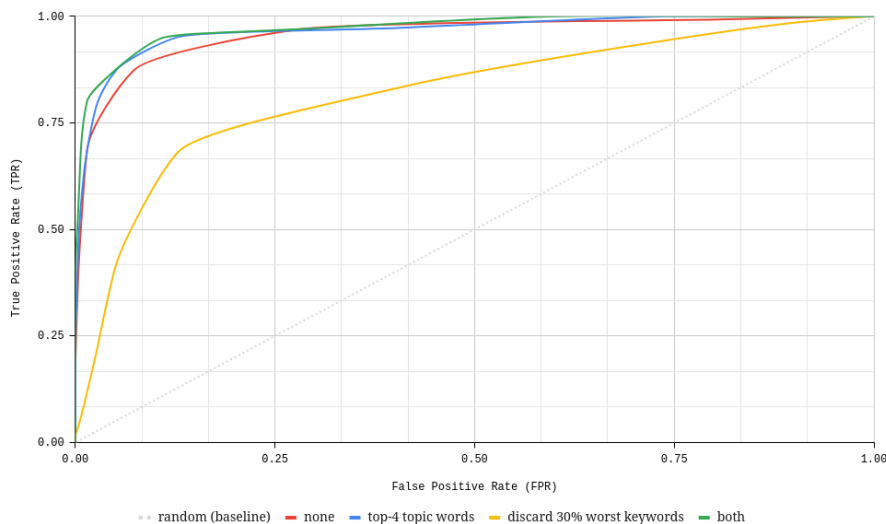


Figure 5.4: ROC curves obtained, using the word2vec model, with the different heuristics.

Analyzing the results, the combination of both heuristics appears to offer the best classifier. A relevant observation is that the keyword heuristic on its own significantly lowers the classifier’s quality, even when compared to the classifier with no heuristics. Nevertheless, it slightly improves the performance when combined with the top four-word heuristic. To better understand these differences, the AUC values for the different scenarios were calculated and can be seen in Table 5.7.

Table 5.7: Area Under the Curve (AUC) values for the different combinations of heuristics.

Heuristic	Area Under the Curve (AUC)
none	0.977
top 4 topic words	0.978
discard 30% worst keywords	0.871
<b>both</b>	<b>0.983</b>

To provide an explanation for the reason why these heuristics work, it is essential to reiterate that topic modeling is not infallible. Its results greatly depend on the number and quality of the



texts it receives. Furthermore, in most applications where it is used, the scope is substantially more narrow, allowing the model to be optimized for the type of data it will receive. However, in the context of this work, it must provide good results while remaining agnostic of the topic or breadth and quality of data it will receive. For this reason, the heuristics help attenuate some of the limitations of this approach.

For example, for each stable topic found, the ten most probable terms are kept and later used for calculating the cosine similarities against the given keywords. Out of these ten terms, the first heuristic only considers the top four scoring ones. Intuitively, the reason why this improves the classifier is that if the topic terms reflect the desired topic that the keywords represent, then selecting only the top four scores should inflate the score significantly since, in the original ten terms, it is rare that the model does not pick up some that often appear in the context of the topic but are too general to provide a meaningful similarity score. Conversely, when the topic terms are unrelated to the topic from the provided keywords, the difference between considering just the top four and all of them is expected to be minimal since there is no particular relation. In short, when considering only the top four words, positive cases are expected to significantly increase their scores, while false cases should only be subject to a marginal increase. This means that the difference between the two becomes, on average, higher, and as such, the classifier can more easily distinguish between the two.

The explanation for the heuristic of discarding the bottom 30% scoring keywords is similar. Although, in this case, the error it is trying to minimize is primarily a result of human differences in perception and behavior. It may be the case that the keywords given are related to the desired topic, but some may not see that reflected in their similarity scores. This could be because of a misperception by the user of how related the keyword is. Alternatively, a keyword can also be used extensively in other domains where the word embedding model was more subjected to during training. In any case, filtering out these cases should help avoid hefty penalties to the scores in positive cases while having a small effect in negative ones. Perhaps because the keywords were carefully chosen to reflect the topic as best as possible, the classifier with just this heuristic has a substantially worst diagnostic ability. However, it appears to have some merit in its reasoning when combined with the previous heuristic.

#### 5.2.2.5 Threshold

For the definition of a threshold value, the dataset was first divided into two, 80% would be used for establishing the threshold, and the other 20% would be used for validation, that is, to confirm that the threshold would yield results as expected. The division was made separately for each of the existing topics, so roughly the same proportion of topics was maintained in both train and validation sets.

In Figure 5.5 (p. 52), the ROC curves for both the test and validation dataset are plotted. Both curves have a similar shape, and the corresponding AUC values are 0.986 and 0.975 for the test and validation datasets. Given their similarities, one can assume that defining a threshold using the

test dataset to obtain a certain outcome will yield similar results when compared to the validation dataset.

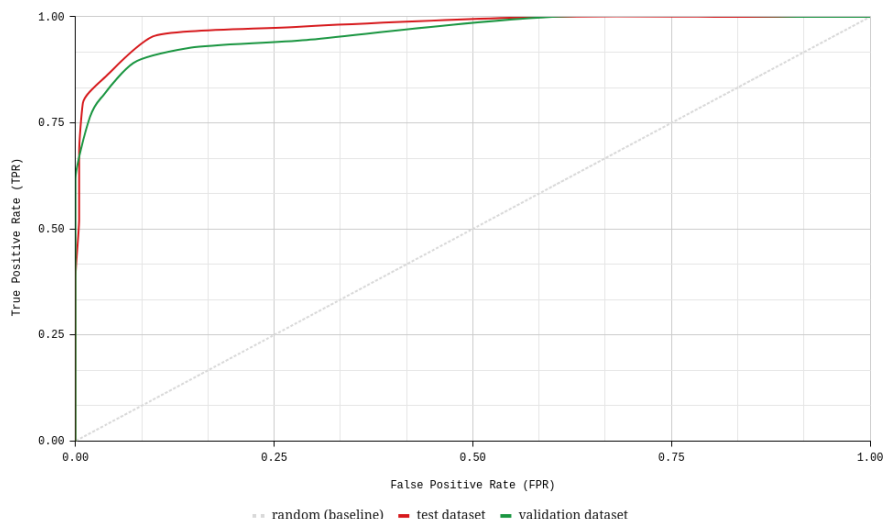


Figure 5.5: ROC curves obtained for the test and validation datasets.

To gather more information about the predictive capabilities of the algorithm, the F1-score and Accuracy were plotted for both datasets. The F1-score is also a measure of accuracy and can be understood as the harmonic mean of the precision and recall (or True Positive Rate). Figure 5.6 (p. 53) shows the different scores for different threshold values. The maximum accuracy and F1-score are reached for both datasets with threshold values between 0.20 and 0.25.

Before defining a threshold value, it is essential to reflect on what exactly is the desired outcome and how it can affect the results later on in the application. Generally, the trade-off when choosing a threshold value is between the number of false positives and the number of false negatives. If it is more important to detect as many positive cases (lower the number of false negatives), the threshold is set lower and will incur an increase in the number of false positives. On the contrary, if there is a higher cost in falsely identifying cases (lower false positives), the threshold is set higher, resulting in more false negatives. For this situation, there is a greater interest in keeping the number of false positives low than in being able to detect most of the positive cases. This is because each profile that gets selected will incur a high computational and memory cost to gather the necessary information, so having more false positives means more processing for information that will not be useful.

Additionally, there is an important characteristic in the similarity metric and how it is calculated that does not apply to most binary classification problems. In many classification problems, it is of little relevancy how far a given score is from the threshold. If the threshold is set to 0.5, an instance having a score of 0.6 and another of 0.9 are viewed the same, as long as the classifier accurately distinguishes the positive from the negative cases. However, in this case, the goal is to gather profiles related to a given topic so that later on, information can be extracted from them,

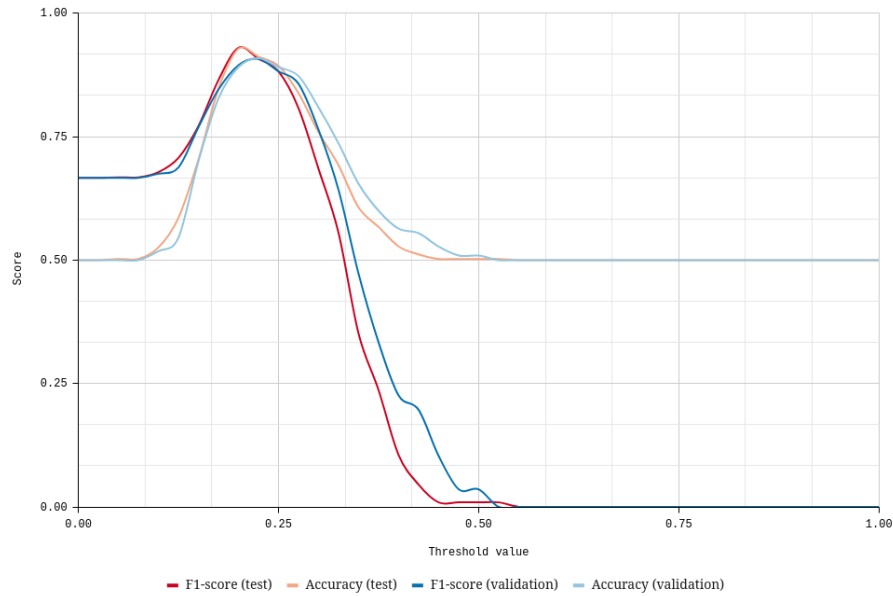


Figure 5.6: F1-score and accuracy as a function of the threshold value for test and validation datasets.

where it is likely that the most pertinent information is necessarily the one related to the topic. As such, in a way, the similarity metric can also be viewed as a predictor of how likely it is for a given profile to contain relevant information regarding the desired topic. If a profile actively and frequently shares content related to the topic, the higher its expected similarity will be and the more likely it will be to share data linked to the topic. This is to say that higher similarity scores are probably correlated with more useful profiles, and as such, setting a higher threshold means not only that it is more likely that the profile is indeed related to the topic but also that the accepted profiles are more likely to share relevant information and be more useful.

Taking this into account, a threshold value of 0.275 was chosen. According to the dataset, it should result in about an 80% true positive rate, so 8 out of 10 related profiles will be selected while only having around a 3% false positive rate. Once again, it is likely that the 20% of related profiles that are not identified do not engage with the topic enough on Twitter, so the quality and quantity of the data that would be able to be retrieved is expected to be low. Nevertheless, this threshold can easily be modified in the future to accommodate other scenarios and objectives where a high recall is more desired.

### 5.3 URL Extraction & Crawling module

The URL Extraction & Crawling module follows up after the initial set of profiles has been filtered, containing only those classified as being sufficiently related to the user's desired domain.

The module is responsible for taking the links collected for each profile, crawling the corresponding websites, identifying which ones are related to the profile in question, and, if it is the case, collecting and structuring information available on the websites.

In Section 5.3.1, the module's process flow is detailed, together with an explanation of the algorithms developed.

Like the Profile Selection module, the approach taken when developing this module was also based on experimentation. More specifically, experiments testing the classifier's ability were done to corroborate the algorithm and inform the choice of a threshold.

Section 5.3.2 (p. 61) documents the creation of the dataset necessary to conduct the experiments, followed by their results and how they shaped the final module's algorithm.

### 5.3.1 Process Flow

Like the previous two modules, in order to work, the URL Extraction & Crawling module needs a few configuration parameters. More specifically, it must first get the following two fields from the configuration:

- **links\_per\_profile** - the maximum number of links to crawl per each profile, not including the ones crawled in depth, that is, reached from an original link;
- **entities** - a JSON containing the desired categories, taken from Spacy's NER, for named entity extraction from the crawled websites, which can be any subset of the following:
  - *person* - includes real and fictional people;
  - *norp* - includes nationalities, religious and political groups;
  - *fac* - includes buildings, airports, highways and other man made structures;
  - *organization* - includes companies, agencies, institutions and other organizations;
  - *location* - includes countries, states and cities;
  - *places* - includes other locations and landmarks, like mountain ranges and bodies of water;
  - *product* - includes any product that is not a service;
  - *event* - includes natural and human-made events;
  - *art* - includes titles of works of art, be it books, songs, paintings, among others;
  - *law* - includes named documents that were made into laws;
  - *language* - includes any named language;
  - *date* - includes absolute and relative dates or periods;
  - *time* - includes times shorter than one day;
  - *percent* - includes percentages;
  - *money* - includes monetary values with the respective units;

- *quantity* - includes any measurements;
- *ordinal* - includes ordinal numbers;
- *cardinal* - includes all other numerals;

The whole process that happens in the URL Extraction & Crawling module can be subdivided into three stages: (1) Crawling; (2) Classification; (3) Entity Extraction.

All links gathered from a Twitter profile in the previous module are first put into a queue. Following it, multiple threads are spawned, where each will remove a URL from the queue, and perform each stage. This is done until the queue is empty or the maximum number of links per profile is reached.

The advantage of using threads instead of new processes to parallelize the processing of multiple URLs is that they are more lightweight than processes while still addressing the bottleneck in this module.

After some testing and measurements, it was found that in each URL processed, the most significant time sink was performing the requests to get the webpages. In fact, while data extraction, classification, and entity extraction lasted less than one second for most cases, requesting the web content took anywhere from two seconds to two minutes. The reason behind this is twofold. First, some links are downloadable content, like files or executables, which take significant time to transfer everything. Even employing some techniques to ignore these URLs, which will be later explained in Section 5.3.1.1, it is not always possible to avoid them. Secondly, if Javascript is required to crawl a website properly, a tool is used that emulates a Chrome browser, which is a heavy, time-consuming process. Thankfully, these time-sinks are mostly I/O bound, meaning that threads offer a solid improvement since when one thread is waiting for a response from the server, another thread can advance with its processing. Moreover, threads do not have such processing overhead to create them, and sharing data between threads is simpler than with processes.

#### 5.3.1.1 Crawling phase

The principal focus of the **Crawling** phase is to gather data from websites that can be used to identify if the link is related to the profile but also includes data that can be useful for the user.

This phase is essentially a specialized web crawler tailored to the application's needs. More specifically, with each URL, it will do a depth crawl, that is, continue crawling using the links found on a webpage, but always staying within the same hostname. While crawling, it will try to identify and collect the following information:

- **names** - these are the values from the “*name*” attribute sometimes found on JSON-LD <sup>7</sup> scripts;
- **titles** - these can include: text found in the <title> HTML tag; values from the “*headline*” attribute present in JSON-LD scripts; or the content found in the “*title*” prop from <meta> HTML tags;

---

<sup>7</sup>JSON-LD - [json-ld.org](https://json-ld.org)

- **description** - content found in the “*description*” prop from <meta> HTML tags;
- **keywords** - content found in the “*keywords*” prop from <meta> HTML tags;
- **internal links** - all URLs that have the same hostname as the original link;
- **external links** - any URL found that is not detected as being an internal link;
- **emails** - the “*href*” strings tagged with “mailto:” found in <a> HTML tags;
- **phone numbers** - the “*href*” strings tagged with “tel:” found in <a> HTML tags;
- **images** - these are links to images, and other information about them, that can be found on: <link> HTML tags when used for image assets; <img> HTML tags; or the “*image*” attribute present in some JSON-LD scripts;
- **corpus** - all the webpage content except for <script> and <style> HTML tags’ content;

After collecting this data from one webpage, if the maximum crawl depth, currently set to five, has not yet been reached and there are still internal links, it will recursively call the crawl function for each of the internal links and concatenate the data gathered into a single document.

The crawl function begins with an HTTP request to obtain the webpage content. Before making the actual request, it will attempt to check if the URL is for downloadable content. This is done through a simple regular expression to look for common file extensions in the URL (*e.g.*, *pdf*, *gif*, *exe*). Although this does not detect every link for downloadable content, it provides an inexpensive (*i.e.*, that does not require making a request) way to filter some of them. If it passes the test, a GET request is made, returning the response headers, status code, and HTML body. At this point, the response headers are analyzed to check once more whether the URL was for downloadable content. This is done by searching the “*Content-Type*” header for downloadable formats (*e.g.*, *video*, *application/gzip*, *text/csv*). If any are found, the response and its contents are discarded, ignoring the URL. Unfortunately, this later detection does not eliminate the time cost of requesting the full HTML body of downloadable content, it simply prevents the further processing of the webpage. In an ideal scenario, this could be counteracted by performing a HEAD request instead of GET. This would provide access to the response headers and therefore check the “*Content-Type*” without requesting the full HTML body. The issue is that most webpages do not follow the standard of providing a useful response for HEAD requests, meaning that, in most cases, the response to a HEAD request returns an HTTP error, rendering it unusable. Given the prevalence of this situation, the approach of performing a HEAD request was discarded. Occasionally, the requests also return an error code due to the default “*User-Agent*” for Python’s requests, usually a 403 or 406 error. If these error codes are returned, the request is performed again with a different mock “*User-Agent*”.

After retrieving the status code and HTML contents of a webpage, if the former is not an error code, the HTML is parsed using BeautifulSoup<sup>8</sup>, the standard library for parsing HTML in Python.

The following step involves the extraction of all the links found on the website. For most websites, this simply requires finding all `<a>` HTML tags and their respective `"href"` attribute. However, special attention is given to links with specific hostnames from what will be designated as link-tree websites. **"Link-tree websites"** are websites used for the purpose of sharing multiple links to a person's social media presence. Widely used in the entertainment industry, an artist can, for example, provide the link to their link-tree website, which will contain links to their Spotify, Instagram, TikTok, personal website, or any other related platforms the artist chooses. These websites are beneficial for the application since they tend to contain identifying names and links related to a specific person. Suppose one of these websites is classified as being related to the Twitter profile being analyzed. In that case, a safe heuristic is to assume that every link found within the website is also related to the profile. Albeit useful, link-tree websites, like [linktr.ee](https://linktr.ee) or [linkgenie.co](https://linkgenie.co), tend to have the respective external links more concealed than simply in `<a>` HTML tags. For example, [linktr.ee](https://linktr.ee)'s external links must be extracted from a particular `<script>` tag. [linkgenie.co](https://linkgenie.co) on the hand, converts every external link to an internal proxy link, meaning that, in order to get the actual URL, a request has first to be made to the proxy URL. As such, when a link to one of these link-tree websites is identified, the extraction of external links is tailored to that particular website.

Once all links have been extracted, each one will be parsed. If the link contains `"mailto:"` or `"tel:"` it will be kept as an email or phone number, respectively. Otherwise, it is checked if it is a syntactically valid URL. If so, it is added to the internal links set, if it has the same hostname as the original link, or the external links set, if it does not. If a URL has been seen before, it is not added to avoid duplicates.

Finally, it will try to extract all the data mentioned above from the HTML body, namely, names, titles, description, keywords, and images. This is done by searching the parsed HTML body for the specific tags and extracting the desired attributes. Regarding images, some extra processing must be done. Namely, when possible, the following attributes are kept: the image's alternate text; its width and height; and its specified type. Additionally, images that have the `".svg"` extension or the words *icon* or *logo* in their URL are discarded. Images are stored as JSON since they can have any of the previously mentioned attributes besides their URL. This makes it more difficult to detect duplicate since the same URL but with different attributes are considered different images. To surpass this but avoid storing the image URLs separately or in duplicate, a Bloom Filter tested with the URLs is used instead to discard repeated images quickly without needing much extra memory.

As mentioned before, this crawling process is done for each link previously gathered from the Twitter profile. Some links, more specifically their websites, require Javascript to be enabled in order to be able to crawl them and parse the content correctly. Detecting which websites require

---

<sup>8</sup>Beautiful Soup - [www.crummy.com/software/BeautifulSoup](https://www.crummy.com/software/BeautifulSoup)

this is a difficult task since there is no standard or common protocol on how to communicate this through HTTP requests. So, in order to detect these cases, after a link has been crawled to its full depth or after having hit the maximum one, if it does not contain any additional internal links beside the original one, it is assumed to require Javascript. The reasoning is that, in general, the large majority of websites will have at least a few internal links. However, when it requires Javascript, a page is usually returned warning the user to enable it, which does not contain any other links. This can therefore be exploited to identify these cases, knowing that, occasionally, a website that does, in fact, not have any internal links, but does not require Javascript, will be re-crawled unnecessarily.

Crawling with Javascript enabled works in much the same way as described before. The only difference is that instead of using Python's default request library, it uses Selenium<sup>9</sup> WebDriver to emulate a Chrome browser and then perform requests through the browser with Javascript enabled. It performs the same checks to avoid requesting downloadable content using the standard requests library since using the browser on URLs with downloadable content would actually download the underlying files, which is not desired.

#### 5.3.1.2 Classification phase

After a link is fully crawled and the corresponding JSON containing the extracted information is obtained, the **Classification** phase will use much of the extracted information to attempt to find a relation between the original Twitter profile and the content found through the link. However, since the web pages that were crawled can literally be any website on the Web, few assumptions can be made about the structure of the data that is found.

So, in order to do this classification, besides the extracted data, it will also require the name and username of the Twitter profile. The idea is to try to find matches of either the name or the username in the extracted content but allow for some variation instead of looking for exact matches. For example, if a Twitter profile belongs to a co-founder of some organization, it is expected that their name will appear on the "about" page of that organization's website. Even so, a link to the organization's website home page is more likely to be shared than one to the about page. This is one of the principal reasons why each link is crawled up to a certain depth, with the intent that, even if it starts at a more general page, it has a chance to end up at another, more useful one.

Before searching for a match, the name and username must first be converted to regular expressions. A Twitter name can contain Unicode characters, making the matching process difficult. So, prior to this, the text is Unicode normalized. That is, Unicode characters that have similar ASCII translations are converted. For example, an "ç" is translated to the character "c". Unicode characters without a translation, like emojis, are removed.

Additionally, it is not uncommon for people to add extra information to their names unrelated to their actual or online name. For example, a popular trend is to include a person's pronouns

---

<sup>9</sup>Selenium - [www.selenium.dev](http://www.selenium.dev)



between parenthesis (e.g., “(she/her)”) or to add a hashtag for a cause they support. This is detrimental for the purposes of this classification since it can easily lead to more false positives and negatives. As such, words which are between parenthesis, following a hashtag, or an at character are removed.

We will use an example username to better explain the classification process, such as “\*John\_Doe\*”. The first step is to convert any possible word delimiters to whitespaces. A word delimiter is considered to be any character that is not an alphanumeric character, including underscores. Unnecessary whitespaces are also trimmed afterward, and every character is lowercased. So, in our example, the username would become “john doe”. If the username contained any ampersand (i.e., “&”) characters, besides removing them, another expression would be added with it converted to “and”.

The second step is to create a separate list with each token found in the string. A token is simply a word delimited by whitespace that is at least three characters long. This avoids adding terms like “Dr” that can often appear in text and would skew the score. Searching for each token broadens the ability to find matches but also has the effect of leading to more false positives. Steps to mitigate this will be taken further ahead. Returning to our example, besides the full expression “john doe”, two tokens would also be considered, “john” and “doe”.

The next step is to convert these strings into usable regular expressions, specifically regular expressions capable of fuzzy matching. Fuzzy matching performs matching of regular expressions but allows for some errors in the terms. These errors are defined by the Levenshtein distance between words, that is, the minimum number of single-character additions, removals, and substitutions necessary to change one string into another. To perform regular expression fuzzy matching, special characters must be added to inform the regex matcher of how relaxed the matching should be. Taking our example, “john doe” is converted to:

`“( ?b )(john){e<=1}( \w ){0,2}( \W | _ | \W | _ )( \w )*( \W | _ ){0,2}( \w ){0,2}( ?b )(doe){e<=1}”`

Each term is enclosed in parenthesis and followed by “{e<=n}” which tells the matcher only to accept matches of that term with at most  $n$  errors, be that substitutions, additions, removals, or a combination of them. The value of  $n$  is determined by the length of the term it affects. More specifically, if the length is less than 3,  $n$  will be 0 (i.e., it must be an exact match), if it is between 3 and 5,  $n$  will be 1, and lengths bigger than 5 have an  $n$  of 2.

It must also be preceded by “(?b)” that forces the algorithm to find the best possible match instead of just the first one. Additionally, and unrelated to fuzzy matching, every whitespace is replaced with “( \w ){0,2}( \W | \_ | \W | \_ )( \w )\*( \W | \_ ){0,2}( \w ){0,2}”. This allows for up to two words to be between each of the original expression’s terms, where the words can be delimited by any non-alphanumerical character. The reasoning behind it is that it is not uncommon for middle names to be omitted in usernames, but these can easily appear on related websites. For example, journalists might have their Twitter name as their first and last name, but in an article, they would sign it with their full name.

All expressions and tokens are then converted to regular expressions with the above method. Their original string length is also kept as it will be useful to determine the score later on. An issue

that arises when extending the search to each token is that it is necessarily more likely to find a positive match with a token than with the whole original expression. Furthermore, a match to the whole expression provides significantly higher confidence that the profile and the webpages have a relation than matching just one token. Because of this, each regular expression is associated with a weight that will dictate how much a match influences the final score. More specifically, entire expressions are given a weight of 1.0, and the tokens are given a weight inversely proportional to their frequency, ranging from a maximum of 0.25 to a minimum of 0. To calculate a token's frequency, the Zipf frequency scale from the Python library `wordfreq`<sup>10</sup> is used. The Zipf frequency scale is a base-10 logarithmic scale of how many times a word appears per billion words. So, for example, a word with a Zipf value of 6 tends to appear once every thousand words. The values range from 0 to 8. The higher the value, the more frequent the word is. The dataset used to determine the word frequencies is the Exquisite Corpus<sup>11</sup>, which includes data from multiple sources like Wikipedia, News, Books, Twitter, Web, among others. Taking our example, the token “john” has a Zipf value of 5.38 and “doe” of 3.84. Intuitively this makes sense since John is a far more common name than Doe. The weights for our example, after being normalized to a value between 0 and 0.25, are of 0.082 for “john” and 0.13 for “doe”. A diagram summarizing the process of creating regular expressions from the original name can be seen in Figure 5.7.

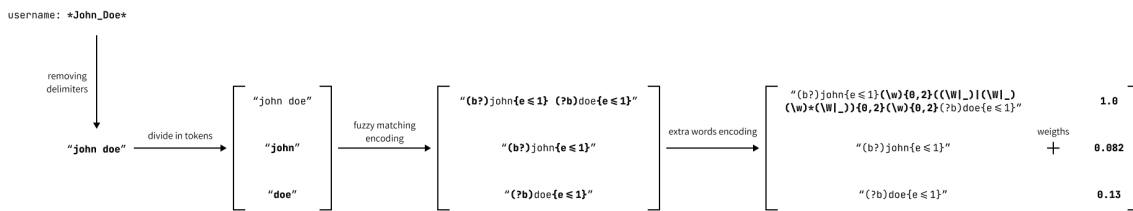


Figure 5.7: Diagram summarizing the process of regular expression formation from an original name string.

From the original Twitter name and username, a set of regular expressions, the original string length, and corresponding weight have been obtained. Following this, each of these expressions are used to search for matches in the data extracted from the webpages, in particular, it will search in the *names*, *titles*, *description*, *keywords*, *internal links*, *external links*, *emails* and the *HTML corpus*, each with equal weight in the final score. Depending on the data field being searched, the expression can be slightly changed to enforce the expression's boundaries. For example, when searching in the set of *emails*, if the expression appears in any email it is probably followed by an “@” symbol and the addresses domain name, so enforcing explicit expression boundaries would not detect most suitable matches. However, if looking for a match in the *description*, the expression should appear bounded by non-alphanumeric characters since otherwise, it could just be part of another, longer word. In this last case, the regular expression is enclosed with the following string, “(\\W|\_|\\b)” to account for this.

<sup>10</sup>wordfreq - [github.com/rspeer/wordfreq](https://github.com/rspeer/wordfreq)

<sup>11</sup>Exquisite Corpus - [github.com/LuminosoInsight/exquisite-corpus](https://github.com/LuminosoInsight/exquisite-corpus)

If a match is not found, the score for that particular search will be 0. If a match is returned, the score for that match is calculated using the following formula:

$$score = \frac{L_{original}}{L_{original} + |L_{original} - L_{match}| + 2 \times n_{errors}} \quad (5.1)$$

Where  $L_{original}$  is the length of the original string,  $L_{match}$  is the length of the match string, and  $n_{errors}$  is the number of errors, *i.e.*, the Levenshtein distance. If it is an exact match, the terms cancel out, and the resulting score is 1.0. Otherwise, the more the match string deviates from the original one, the lower the score will be. Each score is also multiplied by the corresponding regular expression's weight previously calculated.

Finally, the scores from all the searches are added, normalized to a value between 0 and 1.0, and then the average of the scores is taken as the final score. If this score is above a predefined threshold, the link is considered to be related to the Twitter profile, and all the collected information is kept. Otherwise, all data is discarded.

As mentioned above, if a link is for a “link-tree” website and is identified as being related to the Twitter profile, all its external links can also be considered related. If this is the case, each external link is put in the link queue to be processed.

### 5.3.1.3 Entity Extraction phase

To conclude this module's process flow, the **Entity Extraction** stage follows the previous one only if the link was identified as being related to the profile.

It uses Spacy's <sup>12</sup> named-entity recognizer pre-trained module to extract named entities from the HTML corpus collected. It will only save those entities that fall into the categories defined by the user in the configuration parameters. The extracted entities are divided into their respective categories, removing duplicates and appending this information to the link's overall JSON containing the information gathered.

Finally, the HTML corpus is discarded, and the link's processing is completed.

## 5.3.2 Evaluation

Like the previous module, the URL Extraction & Crawling module was tested with a custom built dataset, whose creation is explained in Subsection 5.3.2.1. It was used to validate the use of both the Twitter name and username as a source to create regular expressions, detailed in Subsection 5.3.2.2, as well as the use of the individual word tokens for additional search possibilities as explained in Subsection 5.3.2.3. Finally, it served to empirically set the classifier's threshold value, as described in Subsection 5.3.2.4.

---

<sup>12</sup>Spacy - [spacy.io](https://spacy.io)

### 5.3.2.1 Dataset

The dataset to be built had to associate Twitter profiles, more concretely Twitter handles, to URLs of websites related to the people or organizations behind the profiles. Its creation began using the list of Twitter handles gathered from the previously created dataset, see Section 5.2.2.1 (p. 44). Twitter has a dedicated space in a profile's header to include a personal link. Additionally, if a profile has links in its description, these are automatically detected and parsed by Twitter. A small script was created to iterate through each Twitter handle and request the links mentioned above. Then, after pairs of Twitter handles and links were formed, each was manually revised to assess if the two were indeed related. This involved manually navigating through each website, looking for any reference of the person or organization from the profile.

From the initial 271 profiles previously selected, a total of 325 distinct pairs of Twitter handles and URLs were collected. Not every initial profile was matched to a link, with the final dataset containing 196 different Twitter profiles, some paired with more than one URL.

After having the dataset built, the algorithm was tested to validate some of the decisions made and gauge how they affected its efficacy. Namely, the choice of the initial search expressions, the division of names in tokens, and which threshold was best for separating related from non-related links.

Much like the approach taken in the previous module, the comparison was made by analyzing Receiver Operating Characteristic (ROC) curves in conjunction with the calculation of the Area Under the Curve (AUC) values.

To calculate the TPR and FPR, the match scores were calculated for each link twice. One where it associated the link to its correct Twitter profile, that is, the one that was related to it, and another with a wrong Twitter profile. To isolate the variables being tested, each link was tested against the same wrong profile. In other words, on every test a link  $l_x$  was always tested against the same wrong profile  $p_y$ . The TPR and FPR values were calculated for all threshold values between 0 and 1 with an interval step of 0.025.

### 5.3.2.2 Search Expressions

A Twitter profile can contain multiple identifying information about the person or organization behind it. It can contain the location, a description, and even a link, besides the mandatory name and username. Although all this information could be used to evaluate the relatedness between a website and a profile, not all of it is reliable, nor is it apparent how influential it should be for scoring. For example, the location can be used to provide more confidence in a particular website if, for example, the website references it multiple times. Nevertheless, adding a location is optional, and people can often relocate without remembering to change it. Moreover, how much weight should be given to the location is a complex problem. Some cities have much higher populations or are mentioned frequently, like New York or Los Angeles, so finding them on websites should

not influence the score significantly. Nevertheless, population numbers are not a one-to-one correlation with Twitter profiles. Some major cities in China have large populations, but Twitter is not a common social media in this country, so finding them on websites should carry more weight.

For these reasons, the two attributes tested for the initial search expressions were the profile's name and username. Since these must be defined for every profile, they are a reliable source. Furthermore, the username must be unique, and the names tend to be highly personalized, so finding them on websites should give a strong indication that there is some connection. Nevertheless, the assumptions could be false, and it could be that either one of them or both are not a strong predictor that could be used for the classifier.

In order to verify this, the TPR and FPR were calculated for the scores obtained by using just the username, just the name, and the average when using both. The resulting ROC curves are included in Figure 5.8.

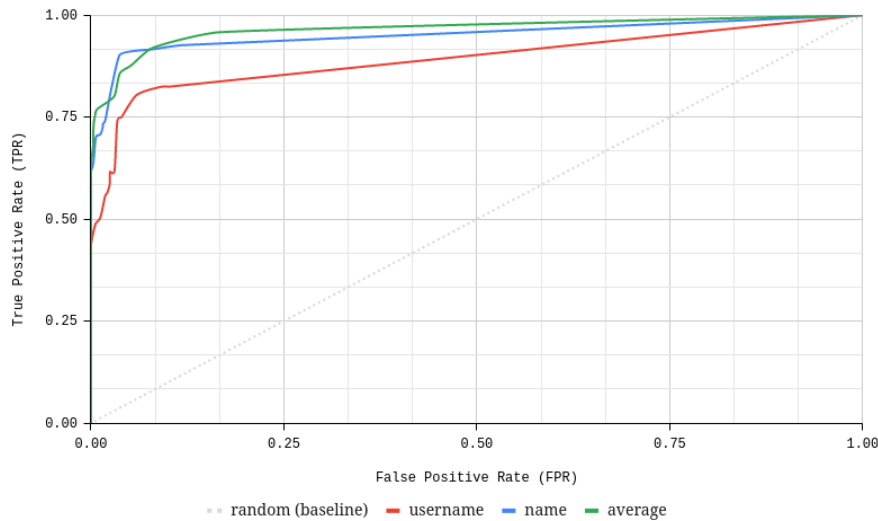


Figure 5.8: ROC curves obtained when using different initial search expression sources.

To aid in comparing the curves, since the differences between them are not significantly pronounced, Table 5.8 contains the corresponding AUC values for each case.

Table 5.8: Area Under the Curve (AUC) values for the different search expression sources.

Expression Source	username	name	average
AUC	0.9726	<b>0.9855</b>	0.9849

Analyzing the results, one can verify that both the username and name can be used as a good predictor of profile relatedness to a given website. The average of both scores appears to perform

slightly worse than just using the Twitter name. Nevertheless, the average was still chosen to calculate the final score. The reason is that a Twitter name, even with the processing described in Section 5.3.1.2 (p. 58), is still likely to contain terms that are not useful for searching. Moreover, unlike the username, the name can be changed at any point, which increases the chance of Twitter users changing it momentarily. As such, it was found that having the Twitter name as the single source of search expressions could make this classifier unable to work for a decent set of users, perhaps not accurately represented in the dataset. Since the difference between the average and just the name is minimal, there is little reason to believe this could drastically worsen the classifier's performance.

### 5.3.2.3 Tokenization

As previously mentioned in Section 5.3.1.2 (p. 58), the algorithm breaks down the initial names into tokens, which are terms separated by word delimiters, that is, any non-alphanumeric character. This broadens the set of possible matches one can find, with the subsequent effect of increasing the chance of non-related matches. Given this, it was necessary to measure how including these tokens affected the classifier's performance. In Figure 5.9 the ROC curves obtained using the classifier with and without tokenization can be visualized.

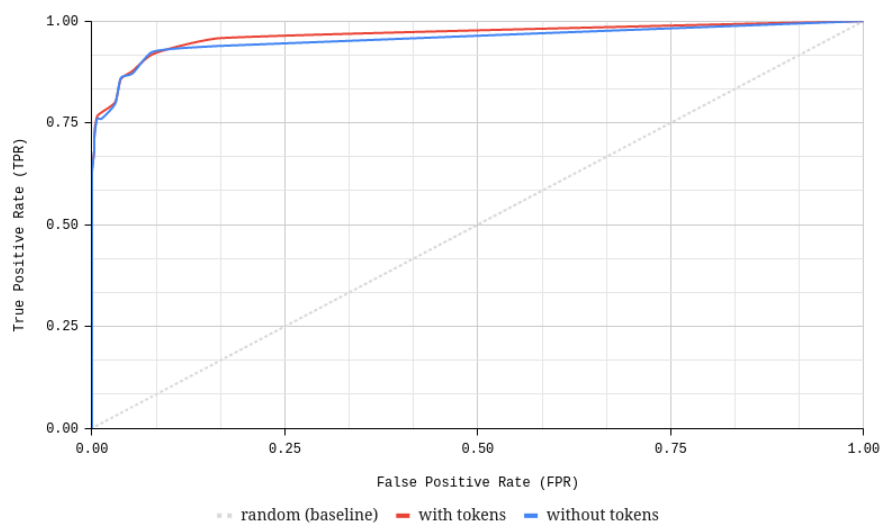


Figure 5.9: ROC curves obtained with and without the search for tokens.

Looking at the figure, the differences between the two are marginal, having the classifier with tokens perform slightly better. This can be perceived best in the AUC values obtained, which are 0.9849 and 0.9812 for the classifier with and without tokenization, respectively. The most probable reason relates to the dataset used and how it was obtained. The links obtained for the dataset were part of the Twitter profile's description, meaning they were likely to be highly related to the profile. For example, many of the profiles for ambient musicians have links to their personal

websites or to their profiles on popular music streaming platforms. This means it is very likely, from a branding standpoint, that either the Twitter name or username is found on these websites. The conclusion is that the classifier primarily deals with simple cases where the inclusion of tokens does not have a substantial positive influence. However, it also does not have a negative effect on the classifier, and therefore, under the assumption that the inclusion of tokens would benefit more in more complex cases, this approach was kept in the final algorithm.

#### 5.3.2.4 Threshold

To establish a proper threshold for the classifier, the dataset had first to be separated into a test dataset, encompassing 80% of the pairs, more specifically 260 of them chosen at random, and a validation dataset with the remaining 20% of pairs. The test dataset functioned to define a threshold which is validated against the validation dataset.

The plot of the ROC curves for both datasets is represented in Figure 5.10. The shape of both ROC curves is similar, with the curve for the test dataset being lower on the Y-axis. These result in the following AUC values of 0.9724 for the test dataset and 0.9788 for the validation dataset. This suggests that the test dataset can be used to define a threshold that will result in a similar performance in the validation dataset.

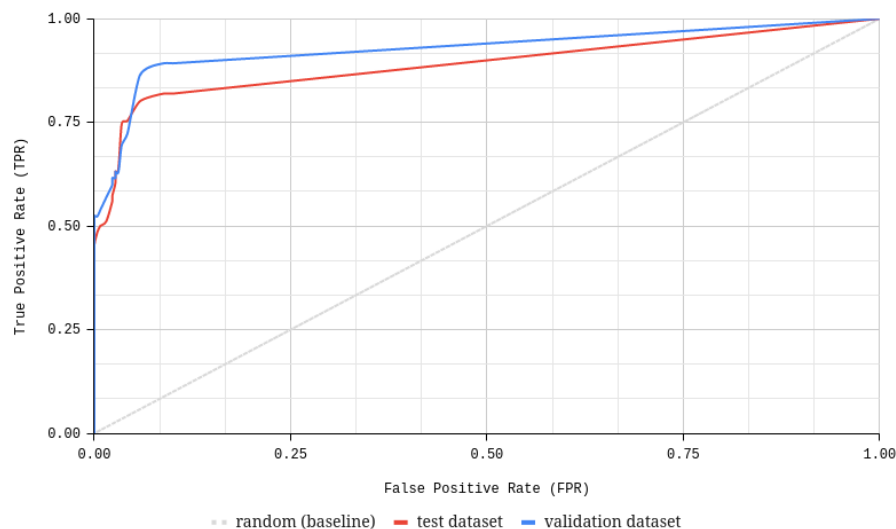


Figure 5.10: ROC curves obtained for the test and validation datasets.

Additionally, the F1-score and Accuracy can be plotted for both datasets for each threshold value to analyze the classifier's diagnostic ability better. In Figure 5.11 (p. 66), one can verify that, in the test dataset, neither the F1-score nor the Accuracy have a peak and simply decrease with the increase of the threshold value. Conversely, in the validation dataset, there is a maximum for threshold values between 0.025 and 0.075, albeit slight.

As stated before, the dataset does not accurately represent many situations where a profile and a website are related, being disproportionately easy than most real-world scenarios. For this reason, although the results invite a threshold value of around 0.07, which the validation dataset predicts would result in close to an 86% TPR with just around a 4% FPR, the final threshold value was set to be slightly less, to 0.06. The assumption is that, with the more common scenarios where the existence of a match on a website is sporadic, the presence of one should be more valued. Hence the threshold, by being lower, catches these instances. It is important to note that, in most cases, a website that is not related will have a score of 0. However, this threshold can easily be changed programmatically to account for new information, whether that be a new, more representative dataset or empirical data.

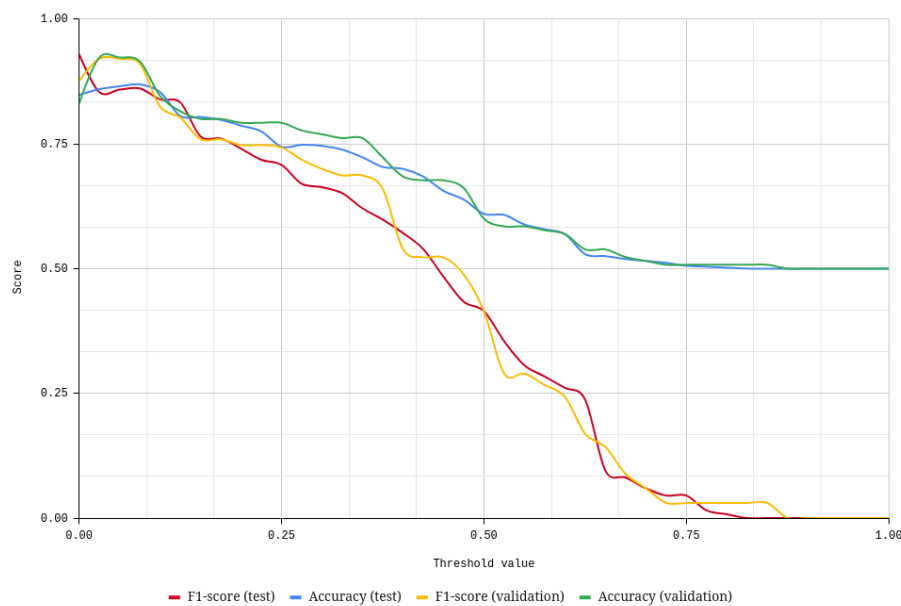


Figure 5.11: F1-score and accuracy as a function of the threshold value for train and validation datasets.

## 5.4 Data Storage and API

The previous modules make up most of the application's logic and are ultimately the components involved in achieving the application's primary goal. Nevertheless, the selected profiles and subsequent data that is collected along the pipeline must be stored so the user can query it. The data must be indexed to allow for the fast retrieval of the information but also allow for full-text search, with capabilities like fuzzy matching, all while remaining scalable with the amount of data. Moreover, since this application has an interface with which the end-user will interact, it must expose a REST API capable of abstracting the logic and serving as the bridge between the interface and the backend together with the database.



Section 5.4.1 details the reasoning behind the chosen database engine and the necessary configurations to use it effectively. Following it, Section 5.4.2 (p. 68) explains the various API endpoints created to allow for a simple interaction with the application's backend by the interface.

### 5.4.1 Data Storage

To meet the previously stated requirements, such as scalability and efficient information retrieval, the Elasticsearch engine<sup>13</sup> was chosen. Besides providing the capabilities mentioned earlier, it also has the benefit of accepting documents in JSON format, which allows for the data to be kept in a Python dictionary while going through the pipeline and then dumped as JSON directly to Elasticsearch, without complex data conversions or insert operations.

To assure that the data is indexed properly and thus allowing for its efficient retrieval and search, an index must first be created with a defined schema that maps each field in the document to its proper types. This will depend on the type of data it stores and how that data is meant to be accessed and used. The schema defined for the documents pertaining to all the data gathered about a profile can be viewed in full at Appendix B (p. 96).

Every field must be given a type that indicates to the Elasticsearch engine the kind of data the field contains and its intended purpose. In the profile data document, most fields are stored as strings and have as their primary type **text**. This type tells the engine to index these fields as *full-text values* meaning that the string value is converted into individual terms, allowing for full-text searching on these fields. This type does not allow for sorting, which can be useful in some instances. If this is the case, extra field types can be added to the same field, making its use possible for multiple scenarios. One common field type added to string values is the **keyword** type, typically used for structured content, that indexes the fields in such a way as to allow for sorting operations. Listing 4 shows the schema for the Twitter profile's ID, described by a text and keyword field.

```
"id": {  
    "type": "text",  
    "fields": {  
        "keyword": { "type": "keyword" }  
    }  
}
```

Listing 4: Elasticsearch compliant schema for the Twitter profile's ID.

Some fields have their own internal document schema as they are made up of multiple other fields, effectively being nested objects. The two examples are the *“processed\_links”* and *“images”* fields. The former is the object that stores the collection of links that were crawled and their

---

<sup>13</sup>Elasticsearch - [www.elastic.co](http://www.elastic.co)

respective data, like “*emails*”, “*internal links*” or “*entities*”. One of these data fields is the “*images*” field that stores the collection of images found while crawling, together with their attributes like “*alt*” text, “*width*” or “*height*”. In order to be able to perform queries on these fields and return them separately from the complete profile document while still being able to filter by fields on the parent object, these must be marked with the type **nested**. The schema for the “*images*” field can be viewed in Listing 5.

```
"images": {
  "type": "nested",
  "properties": {
    "alt": {
      "type": "text",
      "fields": {
        "keyword": { "type": "keyword" }
      }
    },
    "width": { "type": "long" },
    "height": { "type": "long" },
    "src": {
      "type": "text",
      "fields": {
        "keyword": { "type": "keyword" }
      }
    },
    "type": {
      "type": "text",
      "fields": {
        "keyword": { "type": "keyword" }
      }
    }
  }
}
```

Listing 5: Elasticsearch compliant schema for the processed links’ images.

## 5.4.2 API

In order for the interface to access and use the searching algorithm and the data it collects conveniently, it communicates with the backend through a REST API.

This REST API was built using FastAPI <sup>14</sup>, a Python framework that simplifies the process

---

<sup>14</sup>FastAPI - [fastapi.tiangolo.com](https://fastapi.tiangolo.com)

of creating, configuring, and exposing endpoints. The main requirement when building the API, besides allowing the interaction with the application, is that the API should abstract the logic behind most operations as much as possible and require just the necessary information to process the request accurately.

With this in mind, the following five endpoints were created:

- **/search** - triggers a search based on the configurations passed as a JSON body and returns the corresponding search ID;
- **/search/test** - receives just the Profile Acquisition configuration parameters and returns up to 20 tweets obtained from the Twitter API;
- **/searches** - returns all the previous searches metadata, including configuration, current state, duration and errors;
- **/searches/:search\_id** - retrieves, from the Elasticsearch engine, all profile documents collected from the search with ID “*search\_id*” that match a query on the specified fields, both passed as URL parameters. If a query string is not provided, it returns all the profiles collected;
- **/searches/:search\_id/profiles/:profile\_id** - from the profile document with ID “*profile\_id*” collected from the search with ID “*search\_id*”, retrieves all processed links documents that match a given query on the specified fields, or all documents if no query is provided in the URL parameters.

The **/search** endpoint is what triggers the main process to discover and collect data from Twitter profiles related to the domain given in the configurations. Even with the parallelization of operations and the optimizations done to improve performance, this process can take from several minutes to a few hours, depending on the configurations provided. Namely, it depends on the number of initial Twitter profiles, the number of tweets to collect from each profile, and, most importantly, the maximum number of links to crawl for each selected profile.

For this reason, it is impractical for the server to wait until the process is finished to return a response. Many approaches could be taken to resolve this, each better suited for different use cases. The following are the three main solutions that were considered:

- **HTTP Polling** - with this technique, the client polls the server periodically until the process has finished and a response can be given;
- **Server Sent Events (SSE)** - this technique enables the client to automatically receive updates, usually triggered by events, from the server through a one-way HTTP connection;
- **Web Sockets** - this technology is similar to SSE as it allows the client to receive event-driven responses from the server but also permits sending messages from the client to the server through a two-way interactive TCP connection.

Opting for **Web Sockets** would be an overly complex and unnecessary solution since there is no need for a bidirectional connection, as the interface does not update the server after the initial request. Using **SSE** was considered since it could provide the interface with occasional updates on the pipeline's progress, for example, signaling when the Profile Selection module had finished, and it had moved on to URL Extraction & Crawling. Albeit a nice feature, mainly to track the progress in a long session, it is not a viable option. This is because the notion of a linear progression through the pipeline is lost when the modules are parallelized in clusters with Apache Spark. It is possible that while one cluster is actively on the Profile Selection module, another is already in the URL Extraction & Crawling one. As such, there is not an accurate definition of where the overall program is in the pipeline. In turn, this means that the server also does not require sending updates to the client besides the response when the process is finished.

The chosen approach was, therefore, to use **HTTP Polling**. To allow this, each search has a unique ID and a current state that could be: *“running”* if it is still in progress; *“error”* if an error occurred that aborted the search; *“completed”* if it has terminated successfully. The `/search` endpoint triggers the algorithm and returns the generated ID for that search. The interface can then regularly poll the server, using the `/searches` endpoint, and check the status of the search with the provided ID until it is no longer in the *“running”* state.

The `/search/test` endpoint is helpful in allowing the user to test the searching configuration before triggering the main process. The quality of the results greatly depends on the configurations provided by the user. More specifically, the parameters to build the query used for retrieving the initial set of profiles and the keywords selected for the desired domain. Evaluating the quality of the domain keywords *a priori* is difficult since the best would be to test them against a known set of related profiles. Nevertheless, the quality of the initial query can be gauged by collecting a sample of the tweets that would be returned and analyzing them. For example, if the user is interested in the domain of *“tennis”*, one of the provided search keywords may be *“tennis tournament”*. However, suppose from a sample of tweets the user notices that some mention *“table tennis tournament”*. In that case, they can choose to include an exclusion word for *“table”* to avoid processing profiles in the domain of *“table tennis”*.

Regarding the `/searches/:search_id` endpoint, it can receive a query string to search for profiles that match it on some of its fields. By default, it searches in any of the following fields: *username*; *name*; *location*; *description*; and *entities*. However, the endpoint also accepts a string with any subset of these fields separated by a comma, restricting the search to only the provided ones. So, for example, the following request `/searches/12345?q=lisbon&fields=location,entities` will search for profiles collected in the search with ID 12345 where the term *“lisbon”* appears in either the profile's *“location”* or *“entities”* fields. Furthermore, the resulting query sent to the Elasticsearch engine, see Listing 6 (p. 71), performs searching with fuzzy matching, meaning that a profile with the location as *“lisboa”* would also be returned. The **fuzziness** parameter defines the maximum Levenshtein edit distance, which, by being set to *AUTO*, adapts to the query term's length.

```

"query": {
  "multi_match": {
    "query": "lisbon",
    "fields": ["location", "entities"],
    "fuzziness": "AUTO"
  }
}

```

Listing 6: Example of a profile search query for the Elasticsearch engine.

Finally, the `/searches/:search_id/profiles/:profile_id` endpoint, like the previous endpoint, also accepts a query string and optional fields to search for links that result in a profile that return a match. The fields in this case are the following: *name*; *title*; *description*; *keywords*; *phone\_numbers*; *original\_link*; *internal\_links*; and *emails*. However, the Elasticsearch query for this case is more complex due to the nature of the fields and the fact that the processed links are a nested field of the original profile document.

In order to query the processed links and return just their documents while also filtering the profile documents to just the one with the given profile ID, the query must be divided in two. One is to restrict the results to the wanted profile, and the other is marked as *nested* to perform the actual search on the nested documents' fields.

Unlike the fields from the previous endpoints, searching for matches in fields like *internal\_links* or *emails* requires special consideration. For example, for the following request `/searches/12345/profiles/00001?q=john&fields=title,description,emails`, if the email *john@email.com* had been extracted from a link, it is expected that the document for that link would be returned. However, if the matching was done for *emails* like detailed before, even with fuzzy matching, the email would never be matched since the substring `@email.com` surpasses the allowed maximum Levenshtein edit distance. To account for these special fields, the query must be formed to allow for wildcards. These wildcards, which are included as special characters, signal to the Elasticsearch engine that the match must contain the given query string but allow for extra characters where signaled. More specifically, the wildcard character `*` is added to the query string's start and end, effectively allowing for any number of extra characters before and after it. This way, the email would be matched in the prior example since the query terms are present with some additional characters after it. In Listing 7 (p. 72), the constructed query sent to the Elasticsearch engine for the example request can be viewed.

```

"query": {
  "bool": {
    "must": [
      {
        "query_string": {
          "query": "00001",
          "default_field": "id"
        }
      },
      {
        "nested": {
          "path": "processed_links",
          "query": {
            "bool": {
              "should": [
                {
                  "multi_match": {
                    "query": "john",
                    "fields": [
                      "processed_links.title",
                      "processed_links.description"
                    ],
                    "fuzziness": "AUTO"
                  }
                },
                {
                  "wildcard": {
                    "processed_links.emails": {
                      "value": "*john*"
                    }
                  }
                }
              ]
            }
          },
          "inner_hits": { "size": 100 }
        }
      }
    ]
  }
}

```

Listing 7: Example of a profile search query on the processed links for the Elasticsearch engine.

## 5.5 Interface

The final component of the application is its interface. Through the interface, the end-user can configure and trigger a search and visualize its results. The interface was built using the React framework <sup>15</sup>, and its main purpose was to make the application accessible and easily usable by anyone.

The whole interface is made up of four page types: (1) Search page; (2) Searches page; (3) Search Results page; and (4) Profile Links page.

The **Search** page is the initial page a user sees and contains the form necessary to fill in order to request a search. The form is divided into three sections, *Searching Parameters*, *Discovery Parameters*, and *Extraction Parameters*, one for each set of configurations required by the three modules. The section for the Profile Acquisition module includes a button to request a batch of tweets, using the current configurations by calling the API's `/search/test` endpoint. Doing so opens up a modal with the list of returned tweets that can be explored to get a better idea of the kind of profiles the application will be processing and if any adjustments should be made to the parameters. Additionally, the inputs are validated, and error messages are displayed in case mandatory fields are not provided or the values are invalid. Finally, if the inputs are valid, the user can trigger the search. This will redirect to the next page, the Searches page. An example of the Search page with some inputs filled out can be seen in Figure 5.12 (p. 74).

The **Searches** page allows the user to view every search that has terminated or is ongoing. As shown in Figure 5.13 (p. 75), each search has a card with its ID, the date and time it happened, that is, when it was triggered, and, if completed, its duration. Additionally, each card has two icons. The one on the left represents the search's current state. The icon on the right is a button that will open that search's configuration modal when clicked.

A search's configuration modal shows all the parameters used for that particular search. An example of this modal can be seen in Figure 5.14 (p. 76). Given that the search form is somewhat lengthy, this modal has a button to redirect the user to the Search page but with the form already filled with the prior search's settings to facilitate the reuse of configurations. The user can then change the form as normal or trigger the search immediately.

If a search has been completed without errors, the icon on the left will be of a magnifying glass that will redirect the user to that search's Search Results page when clicked.

As the name suggests, the **Search Results** page displays all profiles collected from a particular search. Similar to the Searches page, each profile is represented as a card with the profile's username, the relatedness score to the searched domain, and the number of related links found for that profile, as can be seen in Figure 5.15 (p. 76). Clicking on the file icon in the bottom-right corner of a profile card opens that profile's information modal.

The modal contains most of the information gathered about a profile, including name, location, description, profile image, the number of related links, a list of links that were found but not processed, and the entities extracted from the Tweets. The interface queries Wikipedia's API for

---

<sup>15</sup>React - [reactjs.org](https://reactjs.org)

### Searching Parameters

---

Number of Profiles:

Search Keywords:

Search Hashtags:

Search Excluded Words:

Search Countries (ISO code):

Languages (BCP-47 identifier):

Start time:

End time:

### Discovery Parameters

---

Tweets per Profile:

Topic Keywords:

### Extraction Parameters

---

Links per Profile:

Include Entities:

☒ People
☒ Groups
☐ Structures
☒ Organizations
☒ Locations
☐ Places

☐ Products
☐ Events
☐ Art
☐ Law
☐ Languages
☐ Dates

☐ Times
☐ Percentages
☐ Money
☐ Quantities
☐ Ordinal Numbers
☐ Cardinal Numbers

Figure 5.12: Example of the Search page form interface.

each entity in search of a related page. If found, that entity will appear as a link redirecting the user to the corresponding Wikipedia page. An example of a profile information modal can be viewed in Figure 5.16 (p. 77).

At the top of the page, the user can utilize a search bar to filter the profiles by the presence of keywords in certain fields. The user can specify which fields the keywords must appear on by using the dropdown to the right of the search bar. If none are selected, the search will look for the keywords in all fields, more specifically, the *username*, *name*, *location*, *description* and *entities*.

Both in the profile card and the profile modal, the count of related links can be clicked to redirect the user to the final page, the Profile Links page.

The **Profile Links** page contains all the links processed for a given profile that were identified as being related to it. Following the previous pages, each link is represented as a card with the URL, the relatedness score to the profile, the number of external links found, and the number of images collected. If a link was identified as a *link-tree* website, a small tree icon at the top right corner signals this. Figure 5.17 (p. 77) demonstrates an example of this page. Like the previous cards, clicking on the file icon at the bottom-right opens a modal, in this case, the link information modal.

In this modal, the user can view all the data that was collected while crawling the given URL, such as *names*, *titles*, *keywords*, *description*, list of *emails*, list of *phone numbers*, its *internal*



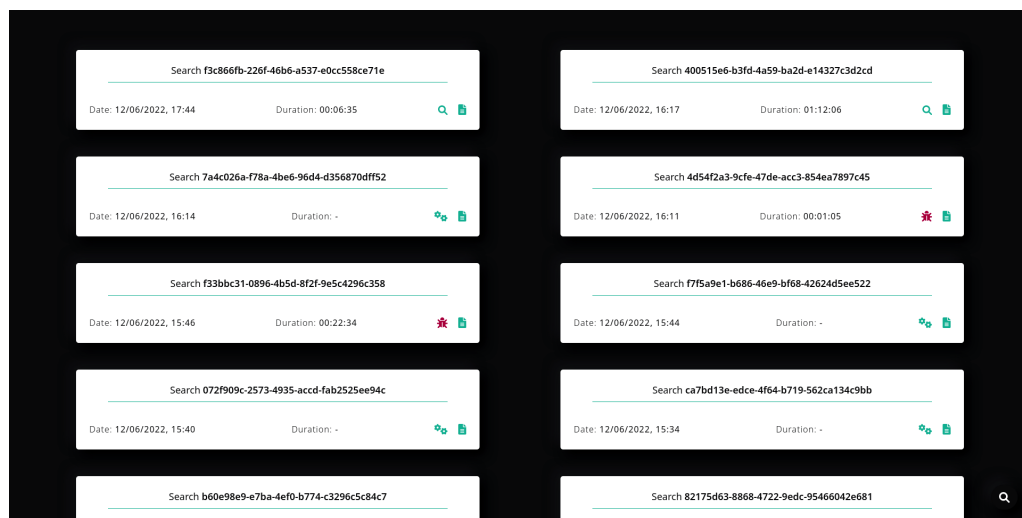


Figure 5.13: Example of the Searches page interface.

*links* and *external links*. If the user requested entities to be extracted, a list of the requested categories and corresponding entities found is shown. Lastly, the images that were extracted appear at the end in an image carousel which can be explored. An example of this modal is found in Figure 5.18 (p. 78).

Finally, the Profile Links page also contains a search bar that can be used to look for links that have particular keywords. This search can be restricted to a subset of a link's fields by using the dropdown.

SEARCH c3a6e0a0-37a1-4b72-a0dc-36370359ccd7 CONFIGURATION

×

---

Number of Profiles: 10

Keywords: **tennis match, atp championship, grand slam, tennis tournament, us open, australian open, rolland garros, wimbledon**

Hashtags: **ausopen, usopen, wimbledon, atptour**

Excluded Words: **golf, table, badminton**

Countries: **US, CA, GB**

Languages: **en**

Start Date: **10/04/2022, 00:00**

End Date: **01/05/2022, 00:00**

---

Keywords: **tennis player, tennis, racket, tennis court, grand slam, tennis tournament, ATP, australian open, us open, rolland garros, wimbledon**

Tweets per Profile: **1500**

---

Links per Profile: **20**

Selected Entities: **person, norp, organization, location**

REUSE CONFIGURATION

Figure 5.14: Example of the Search Configuration modal interface.

<div>tennis</div> <div>Description</div>	
<div>Mike Dickson_DM</div> <div>Score: 0.431</div> <div>Related links: 17</div>	<div>Statsmanuk</div> <div>Score: 0.345</div> <div>Related links: 1</div>
<div>UACoachMurphy</div> <div>Score: 0.331</div> <div>Related links: 0</div>	<div>pavyg</div> <div>Score: 0.326</div> <div>Related links: 5</div>
<div>PlayMassGolf</div> <div>Score: 0.307</div> <div>Related links: 17</div>	<div>CharlotteBSB</div> <div>Score: 0.306</div> <div>Related links: 0</div>
<div>twicktigersfc</div> <div>Score: 0.294</div> <div>Related links: 6</div>	<div>DawgsSports</div> <div>Score: 0.276</div> <div>Related links: 5</div>

Figure 5.15: Example of the Search Results page interface.

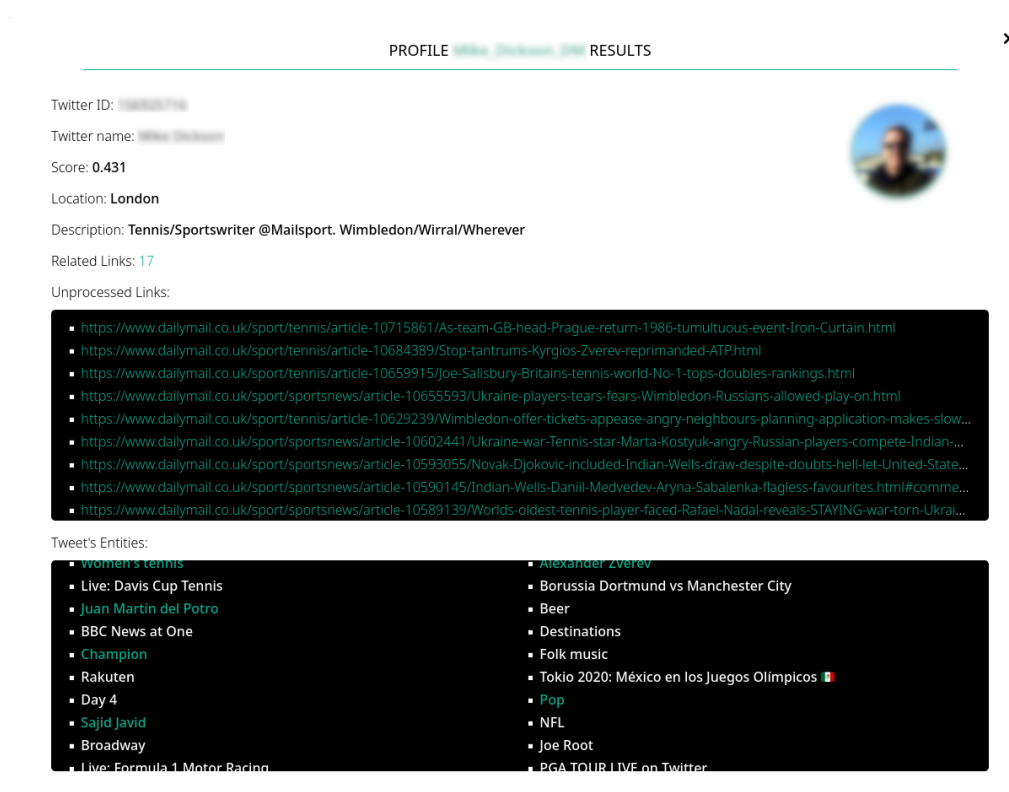


Figure 5.16: Example of the Profile Information modal interface.

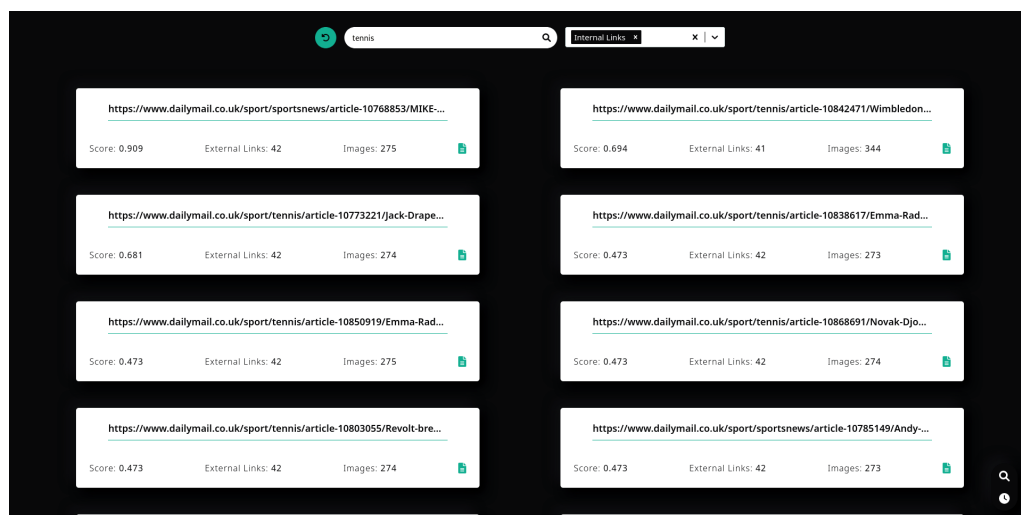


Figure 5.17: Example of the Profile Links page interface.

LINK <https://www.dailymail.co.uk/sport/tennis/article-1084247...> INFORMATION

Names:

- linklist-puff-65598509

Titles:

- Wimbledon has become 'exhibition tournament' after ranking points were taken away, says Cam Norrie Wimbledon has become 'exhibition tournament' after ranking points were taken away, says Cam Norrie
- John Lloyd didn't want his prostate cancer surgery to stop him performing in the bedroom John Lloyd didn't want his prostate cancer surgery to stop him performing in the bedroom
- Federer watches Spanish Grand Prix qualifying in Mercedes garage alongside Toto Wolff Federer watches Spanish Grand Prix qualifying in Mercedes garage alongside Toto Wolff
- Roger Federer 'planning to make injury comeback at Basel Open' in October Roger Federer 'planning to make injury comeback at Basel

Keywords:

- dailymail
- sport
- tennis
- Cameron Norrie
- Wimbledon
- French Open
- formulaone
- Toto Wolff

Description: MIKE DICKSON IN PARIS: The world number eleven even ventured that a few of the top players might not turn up now that the Championships will be 'almost like an exhibition'. Exclusive interview by James Sharpe: John Lloyd's surgeon thought he was joking. 'You will survive this, John,' he reassured him. At least they had caught the prostate cancer in time. Brits Russell and Hamilton qualified fourth and sixth on the grid respectively as the car improvements made by Mercedes took shape. Latest news on Roger Federer including fixtures, live scores, results and injuries plus Swiss stars appearance and progress in Grand Slam tournaments here. Roger Federer has not played since Wimbledon in 2021 when he lost to Hubert Hurkacz and the Swiss Indoors in Basel would represent the 40-year-old's first ATP Tour event since his injury. Latest news and results from French Open tennis tournament in Paris including Roland Garros updates on dates, draw, fixtures and rankings.

Score: 0.694

Emails: -

Phone Numbers: -

Internal Links:

- <https://www.dailymail.co.uk/sport/football/article-10875935/Argentina-fans-descend-Trafalgar-Square-Finalissima-match.html>
- <https://www.dailymail.co.uk/sport/tennis/article-10457835/MIKE-DICKSON-Rafa-Nadals-Australian-Open-win-reminiscent-triumph-Roger-Federer-2008.html>
- <https://www.dailymail.co.uk/sport/tennis/article-10457279/Australian-Open-Sport-stars-react-Rafael-Nadals-incredible-comeback-victory.html>
- <https://www.dailymail.co.uk/sport/tennis/index.rss>
- <https://www.dailymail.co.uk/home/index.html>
- <https://www.dailymail.co.uk/sport/tennis/article-10554899/Novak-Djokovic-emerges-winner-SHOCK-Dubai-quarter-finals-defeat.html>
- <https://www.dailymail.co.uk/sport/sportsnews/article-10849215/John-McEnroe-Tim-Henman-clash-Wimbledons-ban-Russian-players.html>

External Links:

- <https://twitter.com/hashtag/F1>
- <https://discountcode.dailymail.co.uk/debenhams>
- <https://add.my.yahoo.com/rss>
- <https://discountcode.dailymail.co.uk/ao-com>
- <http://itunes.apple.com/app/maonline/id400442503>
- <https://www.primelocation.com>
- <https://twitter.com/intent/follow>
- <https://www.thisismoney.co.uk>

Extracted Entities:

this year, this years, this day, today, tomorrow, today, two or three years, two years, two year up to six weeks, up to two months, Wednesday, week, weeks, year, year-old, years, yesterday

- Products:** 00:34, 00:35, 03/02/22, 219million, 30/01/22, 30/05/22, 6.8k
- Quantities:** 10 kilometres, 12 miles, 120mph, 14 amid foot, 22/05/22 20:11 tunisian, 23/05/22 19:34 mike dickson, 24/05/22 22:30 mike, 25/05/22 14:42, 26/06/21 22:31, 300 miles, 623 per minute, 7.7m, nine-mile
- Structures:** billie jean king cup, billie jean king cup rafael, roland garros 01/06/22, wimbledon cameron norrie
- Languages:** -
- Cardinal Numbers:** 01:38, 05/05/22, 06/05/22, 1, 1,000, 1.12am, 1.1k, 1.2k, 1.4k, 10, 10/05/22, 100, 104, 10:19, 11, 118, 12, 12/04/22, 125, 129, 13, 13/09/21, 132, 135, 139, 13:58, 14, 14/05/22, 148, 149, 15, 150, 152, 155, 15:51, 15:59, 16, 162, 166, 17, 17/01/22, 173, 174, 175, 176, 177, 178, 179, 17, 18, 19, 19:00, 193, 19:31, 11, 2, 3, 4, 3.5, 20, 20/05/22, 201, 202, 203, 204, 205, 21, 21/05/22, 212, 215, 218, 219, 22

Collected Images (344):

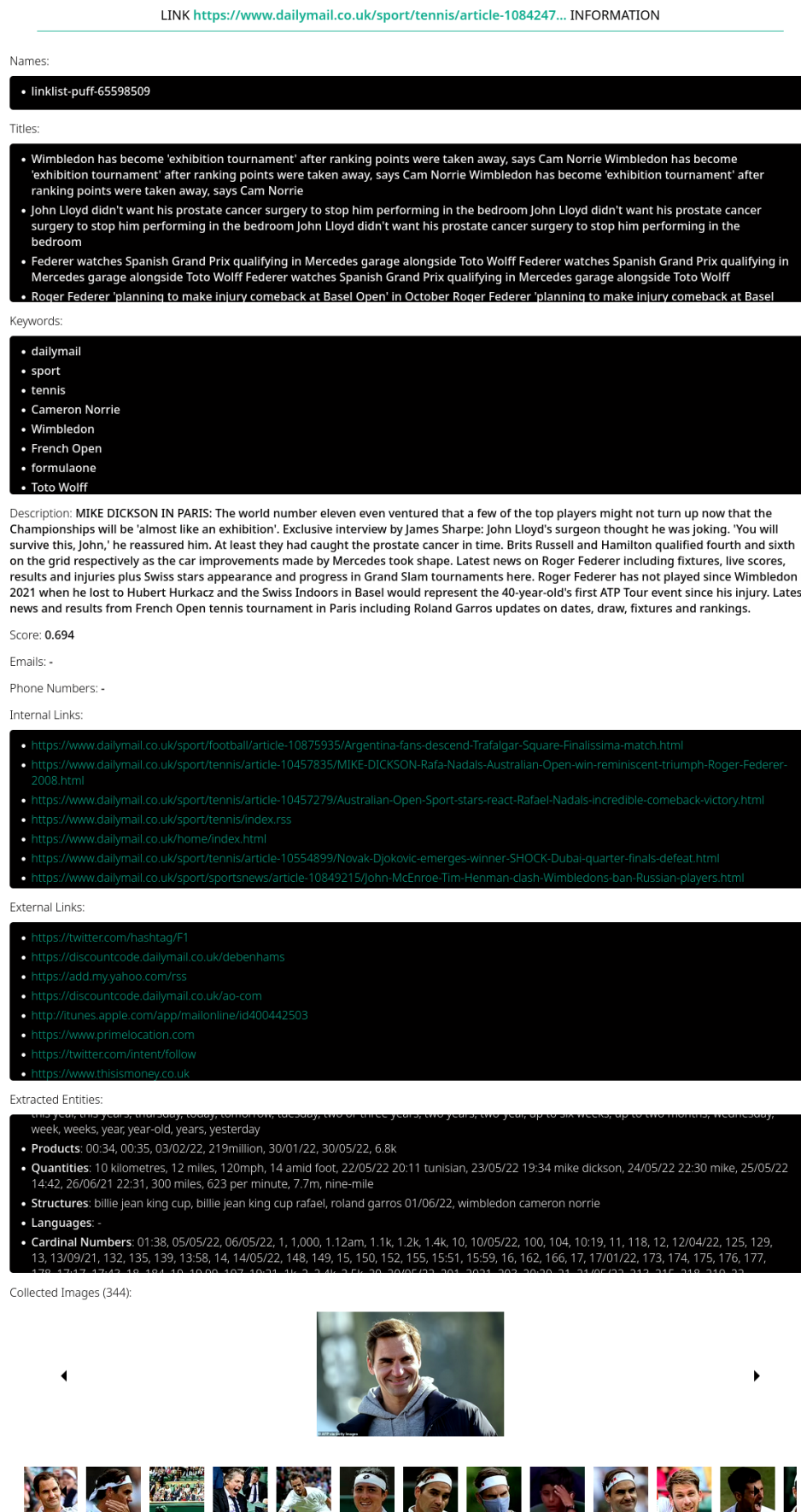


Figure 5.18: Example of the Link Information modal interface.

# Chapter 6

## Case Study in Human Trafficking

---

<b>6.1</b>	<b>Approach . . . . .</b>	<b>80</b>
<b>6.2</b>	<b>Results . . . . .</b>	<b>81</b>
<b>6.3</b>	<b>Conclusions . . . . .</b>	<b>83</b>

---

In order to better understand the capabilities of the developed tool, it is important to use it, mimicking a real-world use case. This allows for an empirical analysis of the application’s efficiency and efficacy and a more explicit baseline of its performance.

The core idea that underpins this work originated from an attempt to facilitate and improve the ability of investigative authorities and NGOs to use the Web as a resource for finding valuable information to combat cases of Human Trafficking and sexual exploitation. Hence, the case study chosen for testing the application is a possible use case that these entities could replicate to discover or get leads on cases of sexual exploitation. The first goal is to identify Twitter profiles of people believed to be practicing sex work. The second is collecting other websites that these people use to promote or distribute their work. For this dissertation, “*sex work*” is defined as the performance or production of sexual services or content for the exchange of money or goods.

It is important to note that by looking to identify people within the sex industry, it does not mean that these people are necessarily committing or being the victim of a crime. In fact, it is expected that most cases are completely innocuous, with the actors involved abiding by their country’s laws. Nevertheless, investigations into Human Trafficking and sexual exploitation require analyzing the sex industry in its entirety to look for suspicious or deviating behavior that can be a sign of a crime since it is an industry with a higher incidence of Human Trafficking victims and perpetrators [38]. Hence, by looking for people within this industry, there is a much greater chance of finding victims of Human Trafficking and sexual exploitation than simply a random set of Twitter profiles.

One of the examples given in Section 1.2 (p. 2) of a platform widely used to sell explicit adult content but with very weak searching functionalities was that of OnlyFans. To attest to the application’s ability to act as a proxy to search OnlyFans profiles and thus serve as a modern

tool for dealing with a still novel platform, more targeted attention is put on finding links to this website.

In general, the set of indicators that are going to be used to empirically judge the application's performance in this particular case study are the following:

- How efficient and effective is it at identifying Twitter profiles from people in the sex industry?
- Is it able to filter which websites are related to the profiles accurately?
- Is it able to collect and associate profiles to the underlying person's OnlyFans account?
- Does it extract information from the websites crawled that can be useful for investigators to discover or extend their knowledge of victims or perpetrators (*e.g.*, emails, phone numbers)?

The rest of this chapter is divided into three sections, Section 6.1 explains the approach that was followed, including the configurations used for searching, while Section 6.2 (p. 81) breaks down the results obtained. Finally, Section 6.3 (p. 83) concludes the results and how they match the initial goals.

## 6.1 Approach

The original approach for testing the application was to request a higher number of initial profiles and configure the searching and keywords to the sex work domain.

The application ran on a Virtual Machine (VM) with 2 AMD EPYC 7502P 32-Core processors, 16GB of RAM, and 70GB of disk space. Unfortunately, the memory limited the case study from being carried out in a single search request. As such, it had to be broken down into multiple searches with practically the same configuration. Specifically, ten separate searches were performed, each requesting an initial batch of 100 profiles for a total of 1,000. It is crucial to keep in mind that, since the searches were done separately, two searches could have processed the same profile. Hence the number of unique profiles that were analyzed is less than 1,000. A reasonable estimate, based on the duplicates from the collected profiles, is between 700 and 800 unique profiles.

Each search's configurations were exactly the same except for the *start\_time* and *end\_time* parameters. Each had different start and end times, the dates varied between the 06/06/2022 and the 12/06/2022, while the time intervals were either 00:00-08:00 or 08:00-16:00.

Regarding the rest of the configuration, all searches requested 100 initial profiles, a maximum of 2,000 tweets to be extracted from each profile, and crawling up to 50 links per profile. Additionally, all entity types were requested, and the searched tweets had to be in the English language and tweeted from within the United States, Canada, or Great Britain.

In order to select tweets likely to be coming from sex workers, with a particular emphasis on OnlyFans creators, the initial search keywords included terms like “escort”, “onlyfans” and

“sex”. Some exclusion words were included to discard tweets with words associated with the previous keywords but unrelated to the desired domain. For example, many of the terms are used by investigators and organizations in the Human Trafficking domain, so words like “*traffic*” and “*trafficking*” were excluded since profiles from these entities are not desired. Additionally, popular media references like “*Sex and the City*”(TV show) or “*The Sex Pistols*”(Musical Band) required also excluding words like “*city*” and “*pistols*”. The full configuration used can be seen in Listing 8 (p. 85).

The terms selected as the topic keywords for the Discovery module were largely similar to the search keywords. Both search and domain keywords should reflect the desired domain. However, while the search keywords can use terms more specifically used on Twitter and can be balanced with excluded keywords, domain keywords cannot and have to use terms that the word embedding model is likely to have been trained with. Hence, specific terms were removed, like “*onlyfans*”, that are unlikely to be well known by the word embedding model. Conversely, other domain keywords were added, such as “*erotic*” which are good descriptors of the desired domain but are often used in the Twitter space in other contexts and, as such, are not apt choices for search terms.

## 6.2 Results

In aggregate, the ten searches lasted a total of 8 hours and 13 minutes, which is roughly 49 minutes per search. An important detail is that the VM used only had two CPUs, meaning that there was no performance gain from the Apache Spark parallelization since it was run in local mode, where each CPU is used as a computing cluster. Since one had to be the orchestrator node while the other functioned as the computing cluster, the execution of the operations was linear.

Each search instance returned, on average, four profiles that it deemed related to the sex work domain for a total of 40 gathered profiles. In reality, because there were some duplicates between searches, as these were independent of each other, the number of unique profiles returned was 31. Considering the total number of profiles analyzed, the application spent about 30 seconds per profile. However, since a good portion of the processing time is spent on crawling and analyzing the websites of related profiles if considering only these profiles, the application spent 12 minutes and 20 seconds per profile. Table 6.1 (p. 82) details the different searches and corresponding metrics from each, including duplicate profiles.

Given the number of the returned profiles, each one was analyzed manually by the author by going through the profile’s posts and related links and evaluating the relatedness to each other and to the sex industry domain.

Out of the 31 unique Twitter profiles returned, 20 appear to be from people who actively perform sex work. From the remaining 11 profiles, 6 were accounts that frequently shared explicit adult content, albeit not produced by them, thus not qualifying as sex workers. However, given the nature of their posts, it is possible that these types of accounts can come from people marketing sex workers or even be undercover pimps, and as such, still be useful to investigations. Another account was concerned with tracking sex offenders, which explains why it was selected. Only in

Table 6.1: Metrics for each search performed for the Case Study.

Date	Time Interval	Elapsed Time	Initial Profiles	Collected Profiles
06/06/2022	00:00 - 08:00	00:47:25	100	4
06/06/2022	08:00 - 16:00	00:54:08	100	6
07/06/2022	00:00 - 08:00	00:43:56	100	4
07/06/2022	08:00 - 16:00	00:47:52	100	4
08/06/2022	00:00 - 08:00	01:00:00	100	7
08/06/2022	08:00 - 16:00	00:51:59	100	2
09/06/2022	00:00 - 08:00	00:46:12	100	3
10/06/2022	00:00 - 08:00	00:48:09	100	3
11/06/2022	00:00 - 08:00	00:45:29	100	3
12/06/2022	00:00 - 08:00	00:47:53	100	4
<b>Total:</b>		08:13:03	1000	40
<b>Average:</b>		00:49:18	100	4

four profiles, no significant relation to the domain at hand was perceived, and are considered false positives, which results in a precision of around 87%. In Appendix C (p. 101), examples of the three types of profiles collected can be found. Table 6.2 breaks down the profiles collected by type together with their corresponding average score.

Table 6.2: Distribution of the gathered profiles.

Type	Number of Profiles	Average Score
Sex Workers	20	0.311
Non-Sex Workers but related to domain	7	0.318
Not related to domain	4	0.315
<b>Total:</b>	31	0.313

Turning the focus onto the links, a total of 137 links were found to be related to the profiles, on average, between four and five links per profile. However, 13 of the 31 profiles did not have any associated links. All 137 links were analyzed to assess the relatedness to their respective profile. It was found that only three did not have any apparent connection and, as such, are considered to be wrongfully classified.

From the 20 profiles considered to be from sex workers, seven had an OnlyFans account link associated, which does not include all profiles that had an account. In analyzing the profiles, a



valuable heuristic for the application would be to consider any link found in the profile's description to be related to the profile. Intuitively this makes sense and would increase the number of OnlyFans links identified to almost all profiles with an account.

Lastly, from the 137 links crawled, 4950 images were identified. Many images are icons, logos, or other assets from the websites which are not particularly useful. However, some that were collected could be relevant in identifying the related people. Additionally, nine emails and three phone numbers were extracted from HTML tags, which can be a valuable resource in investigations for getting leads and identifying known victims. The extracted data breakdown by the previously mentioned profile groups can be found in Table 6.3.

Table 6.3: Distribution of the related links and extracted information by profile group type.

Type	Related Links	Misidentified Links	Emails	Phone Numbers	Images
Sex Workers	132	1	9	3	4824
Non-Sex Workers but related to domain	1	0	0	0	3
Not related to domain	4	2	0	0	123
<b>Total:</b>	137	3	9	3	4950

Other types of collected data, like extracted entities or keywords, are more challenging to analyze in terms of effectiveness without having a specific target in mind. For example, the location can be paramount to filter results when searching for potential victims in a given area.

## 6.3 Conclusions

In terms of *quantitative performance*, given the constraints of the Virtual Machine used for testing the application, it is not easy to assess with certainty how well it would perform when deployed with a proper setup. Taking advantage of the parallelization achievable through Apache Spark, the total compute time could drastically decrease depending on the available computing nodes. Having four computing nodes in use could theoretically cut the total time down to a quarter. With computing clusters becoming cheaper, substantial improvements in performance can be achieved by simply adding more nodes to the system while remaining relatively inexpensive.

The results seem promising when it comes to the *qualitative performance*. The classifier appears to be able to accurately identify profiles from people actively engaging in the sex industry, as well as correctly detect websites related to the profiles. From the results, one can gauge that the tool has a good precision, that is, most results it returns are relevant to the domain. No conclusions can be drawn regarding its recall since we do not know how many relevant profiles and links it processed but was unable to identify accordingly. Nevertheless, it still demonstrates its usefulness if used in Human Trafficking and sexual exploitation investigations. By having good precision, authorities can be confident that most profiles returned are from people performing sex work.

Hence, they can query the profiles and links looking for indicators of suspicious behavior or look for the use of a particular name or location known to be from a potential victim or perpetrator. Furthermore, the association with the respective OnlyFans accounts allows for the exploration of this platform in a way not possible with its current data search limitations.

From the four initial indicators specified, one can conclude that, overall, the application seems effective in identifying Twitter profiles from people in the sex industry, accurately collecting related links, and extracting useful information from them. Its efficiency is undetermined since it is possible that it could be substantially improved with parallelization on multiple clusters. In addition, although it could detect some links to OnlyFans accounts, empirical analysis suggests that there were quite a few profiles with OnlyFans accounts that could have been identified. Nevertheless, as mentioned before, a simple yet seemingly effective heuristic would be to consider all links in a profile's description as being related.

Finally, while working on this case study, a particular tweet drew attention to a profile that, upon closer inspection, appeared to be an underage person offering sexual services, and the account was subsequently reported to Twitter. Albeit an unfortunate occurrence, it is perhaps a testament to the potential of this application. It suggests that, with a proper setup and some minor changes, it could, in fact, be used in a real-world investigation scenario by authorities and organizations to aid in the combat against Human Trafficking and sexual exploitation.

```

{
  "searching": {
    "users": 10,
    "keywords": ["escort", "onlyfans", "camming", "cam girl", "sex",
                 "porn", "stripper", "blowjob", "handjob"],
    "hashtags": ["onlyfans", "escorting", "onlyfansgirl",
                 "eroticentertainer", "stripper", "camgirl"],
    "exclude": ["city", "trafficking", "traffic", "gay", "pistols"],
    "countries": ["US", "CA", "GB"],
    "languages": ["en"]
  },
  "discovery": {
    "keywords": ["sex", "porn", "escort", "erotic", "blowjob",
                 "prostitution", "stripper", "sex work"],
    "tweets_per_user": 2000
  },
  "extraction": {
    "links_per_user": 50,
    "entities": {
      "person": true,
      "norp": true,
      "fac": true,
      "organization": true,
      "location": true,
      "places": true,
      "product": true,
      "art": true,
      "law": true,
      "language": true,
      "date": true,
      "time": true,
      "percent": true,
      "money": true,
      "quantity": true,
      "ordinal": true,
      "cardinal": true
    }
  }
}

```

Listing 8: Example of the configurations used for the Case Study searches.



## Chapter 7

# Conclusions

---

7.1	Conclusions . . . . .	87
7.2	Challenges . . . . .	89
7.3	Contributions . . . . .	89
7.4	Future Work . . . . .	90

---

This final chapter provides a summary of the relevant conclusions drawn from the investigative and development work in Section 7.1. A brief highlight of the challenges faced throughout the process is present in Section 7.2. The contributions achieved with this work are listed in Section 7.3, followed by an outline, in Section 7.4, of possible future work that could be done to improve the application’s capabilities further and extend its applicability.

### 7.1 Conclusions

In a world increasingly present online, the Web holds enormous amounts of information, serving as a reflection of society’s activities. Due to the way it is generated, however, accessing this information using the available technologies is not always trivial. While recognizing the possible misuses of having facilitated access to this information, this work also acknowledges the essential role it can have on good social causes, like journalism and criminal investigations.

To this end, we constructed a hypothesis for a new approach to gathering data pertaining to a given domain by using social media profiles as the starting point. Additionally, we built a tool to test this hypothesis with potential to serve as open-source intelligence for Human Trafficking investigations.

The literature review demonstrated that approaches to forensic technology in Human Trafficking, regardless of their specific goals, relied on data sources that were previously determined to be likely to contain helpful information for Human Trafficking investigations. More significantly, works that fall into the same category as this research rely on crawling multiple websites and gathering large amounts of information, which is dependent on a previously defined list of suspect

websites. Our approach differs from earlier works by using a methodology reliant on Social Media profiles to discover relevant web pages from which meaningful information can be gleaned. Besides not requiring prior knowledge of suspect websites, a potentially notable consequence of following an approach that focuses on social media profiles is that valuable information can be found on web pages unrelated to Human Trafficking activity.

The application developed in this dissertation demonstrates a possible implementation of the proposed approach. Its modular nature also allows ideas from individual components to be gleaned and possibly extended to other uses outside of this work's scope. For example, the concept of using topic modeling allied with word embedding models to identify bodies of text related to a particular domain while remaining agnostic is a valuable tool that could be incorporated into other applications without requiring machine learning approaches.

Additionally, this dissertation also opens up exciting research topics of whose usefulness is potentially significant. An example is the association of websites with people or organizations having minimal information about the entity but even more so about the structure and characteristics of the website. The attempt to perform this association for any website restricts the number of assumptions one can make and encourages the development of creative approaches, some suggested in future work, Section 7.4 (p. 90).

The results obtained in the evaluation of the Profile Selection module, see Section 5.2.2 (p. 44), together with the outcomes from the Case Study, see Section 6.2 (p. 81), indicate that the algorithm designed is able to parse, with good precision, Twitter profiles related to a particular domain given by a set of keywords, which answers the research question RQ1. Similarly, the evaluation of the URL Extraction & Crawling module, see Section 5.3.2 (p. 61), and the Case Study's results demonstrate the ability of the implemented algorithm to measure the relatedness between a Twitter profile and an arbitrary website using only the information that is publicly available from the profile. Furthermore, the module's bottleneck is the inevitable process of performing HTTP requests for each link, with the rest of the classifier having a marginal impact on the overall computing time and therefore answering research question RQ2.

The developed interface, specifically its portion for data visualization and querying, provides a satisfactory answer to the research question RQ3, understanding that further research and performing usability tests with target users would likely showcase areas for improvement.

Due to constraints in development time and available computing resources, an adequate response to the research question RQ4 was not achieved. The application's architecture is designed to easily allow the parallelization of its operations through multiple computing clusters under Apache Spark. Moreover, the choice to use the Elasticsearch engine for data indexing should also guarantee that the querying of the collected information remains possible with low latency, even when massively increasing the amount of data collected. Notwithstanding, this merely provides a theoretical answer to RQ4. A better, empirically-based answer would require testing the application with a setup having multiple computing clusters at its disposal and performing an extensive search, requesting an initial body of profiles in the tens to hundreds of thousands.

## 7.2 Challenges

Throughout this project, some challenges that arose required spending more time on certain stages than initially anticipated. Additionally, some constraints limited the freedom with which things could be accomplished.

The first challenge that spanned the entire work was the relative novelty of the proposed approach. As far as it is known, no other work attempts to collect information scattered around the Web using an approach similar to the proposed one. Although this means that the work has the potential to be more impactful, it also meant that there were few resources, both in the literature as well as in the industry, that mention, let alone tackle, the multiple problems that the application had to solve in order to work. As such, the designed solutions had to be tested and iterated upon without much prior knowledge of what could and could not work. An example of this effort is the two datasets that had to be built in order to test two of the principal modules, as there were no publicly available datasets, to our knowledge, that could be used.

Secondly, many of the tasks done by the application, namely topic modeling and website crawling, are resource-intensive, especially when done on a larger scale. This proved to be a challenge since even a powerful computer does not have the capability to efficiently process as much data as this application would demand in a real-world scenario. This meant that all tests had to be done on a small scale compared to the envisioned use case scenarios.

Lastly, even though the project was approved for Academic Research access to the Twitter API, it still held a limit of 10 million tweets per month. This limit was never hit, but it influenced the testing of the application as tests could not be done without considering possibly approaching the maximum and having the development, in a sense, halted until the following month.

## 7.3 Contributions

The principal contribution that stemmed from this dissertation is the novel approach it puts forth for tackling information retrieval of Web content. This approach and the underlying ideas that were explored hopefully serve as a starting point and inspire more research to be done, leading to better tools for OSINT used for social good.

The resulting application that was developed is also a significant contribution since it not only validates the proposed hypothesis but also can already be used in a real-world investigative setting with minimal changes. As such, all of its source code is available as open-source. The interface code can be found at [github.com/TitoGrine/Thesis\\_Project\\_Interface](https://github.com/TitoGrine/Thesis_Project_Interface), while the application's backend is available at [github.com/TitoGrine/Thesis\\_Project](https://github.com/TitoGrine/Thesis_Project). Additionally, the datasets built and used to evaluate the Profile Selection and URL Extraction & Crawling modules were also made available for general use. Both datasets can be found at [rdm.inesctec.pt/dataset/cs-2022-007](https://rdm.inesctec.pt/dataset/cs-2022-007).

Finally, this dissertation's whole body of work was compiled in a paper with the same title and submitted to *SoGood 2022 - The 7th Workshop on Data Science for Social Good*.

## 7.4 Future Work

With a solid and usable proof-of-concept application, more development and research time would allow for further improvements and expansion of its capabilities. A general yet practical expansion would be to broaden the application's scope to other social media platforms. This would require developing more extensive and better datasets to test each module on new types of profiles.

The classification of a profile's relatedness to a given link could also be expanded. One possible path would be to use named-entity recognition and entity linking to find, in the crawled websites and tweets, aliases of the Twitter name and username. These aliases could then be added to the set of search expressions to match. So, for example, if the Twitter name was "*John Doe*" and in one of the crawled websites a sentence like "*John Doe also known as Jonathan Doe*", the alias "*Jonathan Doe*" could be identified and also searched for as a possible indicator of a website being related to the profile. While analyzing the profiles collected in the case study, one website had an alias of the profile's Twitter name that could have been used to extend the search scope. Another more modern approach would be to train a binary classifier on a dataset of profile-website pairs to identify related pairs. The challenge with this approach would be building a sizable yet representative dataset.

As mentioned in Section 5.3.1 (p. 54), the biggest bottleneck in the application regarding time is the crawling of the URLs, specifically the requests for content. A possible improvement to minimize this could be extending the Spark parallelization to distribute the crawling operations on multiple clusters. This would follow the MapReduce model, where each link would be mapped to the respective profile ID, and after the processing finished, the results would be reduced by key. A downside of this approach would be that the crawling operations would no longer be able to update each other on the links that have already been seen, potentially leading to occasionally double crawling.

Also related to the bottleneck of crawling a profile's gathered links, a maximum number of URLs to crawl per profile is defined in the configuration parameters. This means that some of the URLs collected for crawling go unprocessed if the limit is hit. These links are currently stored, and a future improvement could allow the user to select a profile to continue crawling using the unprocessed links. This could be desirable if a profile is identified as being of particular interest and more information is desired.

The use of the Elasticsearch engine to index the data is ideal when the goal is the fast retrieval of the information together with features like full-text search and fuzzy searching. However, as the platform itself points out, having an engine focused on these features with such low latency means that other, commonly desired properties of database systems are not present. The Elasticsearch engine is built with speed in mind, under the assumption that memory is abundant and not an issue. Although this may change in the future, it currently does not deal with memory errors very well, meaning it is not a *robust* database. Additionally, it does not have any built-in security features. For these reasons, it is usually not recommended for Elasticsearch to be the main database and certainly not the single source of truth of an application's data. This means that another database



should be used that is robust and secure, which then connects to the Elasticsearch engine so the application can function the same. A database like MongoDB <sup>1</sup> would be a great choice since it is also a document-based database that works with JSON and already has plugins made specifically for connecting to Elasticsearch.

To improve the structuring and augmentation of the extracted data, a logical improvement would be to connect the data that is extracted from Twitter profiles and websites with Linked Data from the Semantic Web, for example, with the content present in DBpedia <sup>2</sup>. For this, the data extractor could be enhanced with tools like the Python's HTML Metadata Extractor library <sup>3</sup>. The interface would also need to be updated to allow for a natural exploration of the Linked Data entities in conjunction with the application specific data.

Lastly, to have a real sense of the application's capabilities and its usefulness, the next step would be to deploy it with multiple computing clusters available for parallelization and have it be used in a real-world scenario, perhaps in coordination with authorities, to gather feedback on other improvements that were not apparent.

---

<sup>1</sup>MongoDB - [www.mongodb.com](http://www.mongodb.com)

<sup>2</sup>DBpedia - [www.dbpedia.org](http://www.dbpedia.org)

<sup>3</sup>HTML Metadata Extractor - [usc-isi-i2.github.io/etk/extractors/html\\_metadata\\_extractor.html](https://github.com/usc-isi-i2/etk/extractors/html_metadata_extractor.html)



## **Appendix A**

# **Literature Review Process**

The approach taken for finding the studied literature can be summarized as taking the following iterative process:

1. Given a possibly relevant field for the dissertation, do a preliminary read of its Wikipedia and other web articles to gather keywords and relevant terms;
2. With these terms and considering the purpose within the dissertation, build a set of queries which encapsulate the desired information;
3. Submit the queries and if needed refine them to obtain better results;
4. Prioritizing recent and highly cited results, filter the relevant ones by analysing the title and abstract, confirming its relation to the desired domain;
5. From the relevant documents, perform a light read of the introduction, related work and methodology to gather cited work that also pertains to the domain and could be useful;
6. Taking the collection of cited works, repeat the process from step 4.;
7. If new terms and topics are identified, incorporate them into new queries and repeat the process.

The process can be used to retrieve relevant literature, but it is also possible that from reading the collected works, the original field is found to have little or no usefulness to the dissertation.





## Appendix B

# Elasticsearch Profile Document Schema

```
{
  "mappings": {
    "properties": {
      "id": {
        "type": "text",
        "fields": {
          "keyword": { "type": "keyword" }
        }
      },
      "username": { "type": "text" },
      "name": { "type": "text" },
      "profile_image": {
        "type": "text",
        "fields": {
          "keyword": { "type": "keyword" }
        }
      },
      "location": {
        "type": "text",
        "fields": {
          "keyword": { "type": "keyword" }
        }
      },
      "description": { "type": "text" },
      "entities": {
        "type": "text",
        "fields": {
          "keyword": { "type": "keyword" }
        }
      }
    }
  },
}
```

```
"score": {
  "type": "text",
  "fields": {
    "keyword": { "type": "keyword" }
  }
},
"processed_links": {
  "type": "nested",
  "properties": {
    "description": { "type": "text" },
    "emails": {
      "type": "text",
      "fields": {
        "keyword": { "type": "keyword" }
      }
    }
  }
},
"entities": {
  "type": "nested",
  "properties": {
    "location": {
      "type": "text",
      "fields": {
        "keyword": { "type": "keyword" }
      }
    }
  }
},
"norp": {
  "type": "text",
  "fields": {
    "keyword": { "type": "keyword" }
  }
},
"organization": {
  "type": "text",
  "fields": {
    "keyword": { "type": "keyword" }
  }
},
"person": {
  "type": "text",
  "fields": {
    "keyword": { "type": "keyword" }
  }
}
```

```

    }
  },
  "external_links": {
    "type": "text",
    "fields": {
      "keyword": { "type": "keyword" }
    }
  },
  "images": {
    "type": "nested",
    "properties": {
      "alt": {
        "type": "text",
        "fields": {
          "keyword": { "type": "keyword" }
        }
      },
      "width": { "type": "long" },
      "height": { "type": "long" },
      "src": {
        "type": "text",
        "fields": {
          "keyword": { "type": "keyword" }
        }
      },
      "type": {
        "type": "text",
        "fields": {
          "keyword": { "type": "keyword" }
        }
      }
    }
  },
  "internal_links": {
    "type": "text",
    "fields": {
      "keyword": { "type": "keyword" }
    }
  },
  "is_link_tree": { "type": "boolean" },
  "keywords": { "type": "text" },
  "name": { "type": "text" },

```



```
    "original_link": {
      "type": "text",
      "fields": {
        "keyword": { "type": "keyword" }
      }
    },
    "score": {
      "type": "text",
      "fields": {
        "keyword": { "type": "keyword" }
      }
    },
    "title": { "type": "text" }
  }
},
"unprocessed_links": {
  "type": "text",
  "fields": {
    "keyword": { "type": "keyword" }
  }
},
}
}
```

Listing 9: Elasticsearch compliant schema for profile data documents.



## Appendix C

### Case Study Example Profiles

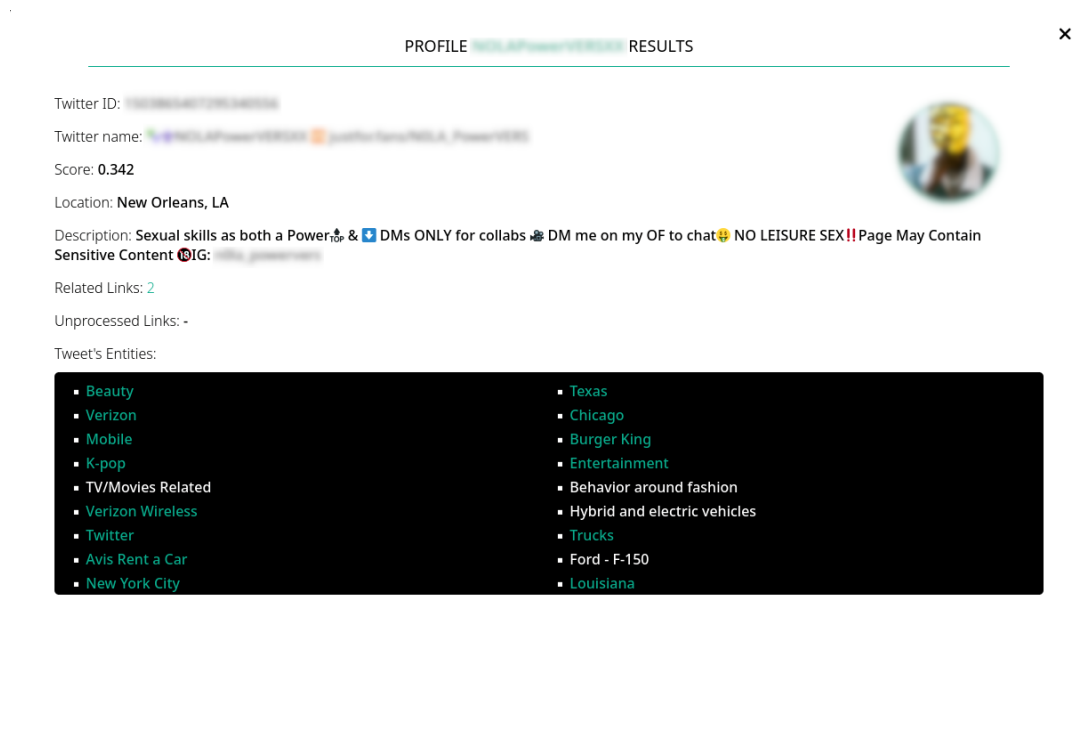





Figure C.1: Example of a profile from a sex worker collected in the case study.

PROFILE [View Profile](#) RESULTS ×



---

Twitter ID: [1440888888888888888](#)

Twitter name: [Sexy Latina Marie Vega](#)   

Score: 0.285

Location: Van,B.C. Canada /FMTY

Description: Montreal Raised Curvy Natural Latina  Sensual/Playful/Sweet / Sexy Experience  Vancouver,Calgary,Winnipeg,Edmonton,Toronto. Saskatoon,Montréal

Related Links: 11

Unprocessed Links: -

Tweet's Entities:


<ul style="list-style-type: none"> <li>Wellness and health</li> <li>Tattoos</li> <li>General Travel</li> <li>Body art</li> <li>People celebrate St. Patrick's Day</li> <li>#HumpDay</li> <li>#TGIF (Thank God is Friday)</li> <li>2月14日はバレンタインデー </li> <li>Easter</li> </ul>	<ul style="list-style-type: none"> <li>Family and life stages</li> <li>Drink Experience</li> <li>Pistachios</li> <li>Nature</li> <li>Travel Actions</li> <li>Fashion &amp; beauty</li> <li>Foundation</li> <li>Drink Novelty</li> <li>Mexican cuisine</li> </ul>
---	--

Figure C.2: Example of a profile from a sex worker collected in the case study.

PROFILE [View Profile](#) RESULTS ×




---

Twitter ID: [1440888888888888888](#)

Twitter name: [Linda R.R. 12](#)

Score: 0.283

Location: Goose Creek, SC

Description:  Charleston, SC   | Owner of [Love And Lust By Me](#) | [Love and Lust](#) provide the very best intimate Bedroom Accessories to make your night CUM-plete

Related Links: 0

Unprocessed Links: -

Tweet's Entities:


<ul style="list-style-type: none"> <li>Mobile</li> <li>Starbucks</li> <li>YouTube</li> <li>Jay Z</li> <li>Careers</li> <li>Butterfinger</li> <li>Squid Game</li> <li>Horoscope</li> <li>Wellness and health</li> </ul>	<ul style="list-style-type: none"> <li>Apple</li> <li>Technology</li> <li>Family and life stages</li> <li>Drink Experience</li> <li>International Pride Day</li> <li>Vegetable recipes</li> <li>21 Savage</li> <li>DMX</li> <li>Personal finance</li> </ul>
--	---

Figure C.3: Example of a profile from a non-sex worker but related to the sex industry collected in the case study.

PROFILE
RESULTS
X

---

Twitter ID:   
 Twitter name:   
 Score: 0.290  
 Location: St Louis, MO



Description: CA girl in the Midwest | Cat mom | Artist | Abortion Activist | Houseplant Addict | Foodie | Karaoke Fan | She/They | Gemini | II ☉  
 » » » ↑

Related Links: 2

Unprocessed Links:

- https://podcastreview.org/review/lolita-podcast
- https://youtu.be/sglCS2Yj6I8
- https://youtu.be/LHlnQQelPIM

Figure C.4: Example of a profile unrelated to the sex industry collected in the case study.



# References

- [1] Defense Advanced Research Projects Agency. Memex. Available at <https://www.darpa.mil/program/memex>. Accessed in December 2021.
- [2] Simon Andrews, Ben Brewster, and Tony Day. Organised crime and social media: Detecting and corroborating weak signals of human trafficking online. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9717:137–150, 2016.
- [3] Clive Best. *Open source intelligence.*, pages 331–343. 2008.
- [4] David M Blei, Andrew Y Ng, Jordan Edu, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.
- [5] Ben Brewster, Timothy Ingle, and Glynn Rankin. Crawling open-source data for indicators of human trafficking. *Proceedings - 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC 2014*, pages 714–719, January 2014.
- [6] Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29:1157–1166, September 1997.
- [7] Tao Chen, Damian Borth, Trevor Darrell, and Shih-Fu Chang. Deepsentibank: Visual sentiment concept classification with deep convolutional neural networks. October 2014.
- [8] DataReportal. Global social media stats. Available at <https://datareportal.com/social-media-users>. Accessed in January 2022.
- [9] DataReportal. Twitter stats and trends. Available at <https://datareportal.com/essential-twitter-stats>. Accessed in February 2022, November 2021.
- [10] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41:391–407, 1990.
- [11] Jacob Devlin, Ming Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference*, 1:4171–4186, October 2018.
- [12] Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. *IJcAI*, 7:1606–1611, 2007.

- [13] Luca Giommoni and Ruth Ikwu. Identifying human trafficking indicators in the uk online sex market. *Trends in Organized Crime*, pages 1–24, September 2021.
- [14] Yaakov HaCohen-Kerner, Daniel Miller, and Yair Yigal. The influence of preprocessing on text classification using a bag-of-words representation. *PloS one*, 15, May 2020.
- [15] Myriam Hernandez-Alvarez. Detection of possible human trafficking in twitter. *Proceedings - 2019 International Conference on Information Systems and Software Technologies, ICI2ST 2019*, pages 187–191, November 2019.
- [16] Thomas Hofmann. Probabilistic latent semantic indexing. *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 1999*, pages 50–57, August 1999.
- [17] Thomas Hofmann. Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning 2001* 42:1, 42:177–196, 2001.
- [18] Liangjie Hong and Brian D. Davison. Empirical study of topic modeling in twitter. *SOMA 2010 - Proceedings of the 1st Workshop on Social Media Analytics*, pages 80–88, 2010.
- [19] Matthew Honnibal and Ines Montani. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing, 2017.
- [20] Michelle Ibanez and Rich Gazan. Virtual indicators of sex trafficking to identify potential victims in online advertisements. *Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2016*, pages 818–824, November 2016.
- [21] D. Jurafsky and James H. Martin. *Speech and Language Processing*, volume 1. 3 edition, 2000.
- [22] L. V. Kapustina. Digital footprint analysis to develop a personal digital competency-based profile. *Lecture Notes in Networks and Systems*, 133:591–596, 2021.
- [23] Andrew Keiper and Perry Chiaramonte. The 'village' of law enforcement agencies battling human trafficking. *Fox News*, May 2019.
- [24] Mayank Kejriwal, Jiayuan Ding, Runqi Shao, Anoop Kumar, and Pedro Szekely. Flagit: A system for minimally supervised human trafficking indicator mining. *Workshop on Learning with Limited Labeled Data*, December 2017.
- [25] Mayank Kejriwal and Pedro Szekely. An investigative search engine for the human trafficking domain. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10588 LNCS:247–262, 2017.
- [26] Craig A. Knoblock, Pedro Szekely, José Luis Ambite, Aman Goel, Shubham Gupta, Kristina Lerman, Maria Muslea, Mohsen Taheriyani, and Parag Mallick. Semi-automatically mapping structured sources into the semantic web. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7295 LNCS:375–390, 2012.
- [27] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive data sets*. Cambridge university press, 2020.



- [28] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10:707, 2 1966.
- [29] Yung Shen Lin, Jung Yi Jiang, and Shie Jue Lee. A similarity measure for text classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 26:1575–1590, 2014.
- [30] Shuhua Liu and Patrick Jansson. City event identification from instagram data using word embedding and topic model visualization. 2017.
- [31] C. Mattmann, G. Yang, H. Manjunatha, Thamme Gowda, A. Zhou, Jiyun Luo, and L. McGibney. Multimedia metadata-based forensics in human trafficking web data. 2016.
- [32] Diego Maupomé and Marie-Jean Meurs. Using topic extraction on social media content for the early detection of depression. *undefined*, 2018.
- [33] Andrew Kachites McCallum. Mallet: A machine learning for language toolkit, 2002.
- [34] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, January 2013.
- [35] Christopher E Moody. Mixing dirichlet topic models and word embeddings to make lda2vec. May 2016.
- [36] Feng Niu, Ce Zhang, Christopher Re, and Jude Shavlik. Deepdive: Web-scale knowledge-base construction using statistical learning and inference. *VLDS*, 12:25–28, 2012.
- [37] United Nations Office on Drugs and Crime. Human-trafficking. Available at <https://www.unodc.org/unodc/en/human-trafficking/human-trafficking.html>. Accessed in November 2021.
- [38] United Nations Office on Drugs and Crime. *Global Report on Trafficking in Persons 2020*. 2020.
- [39] National Center on Sexual Exploitation (NCOSE). A look into onlyfans: Child sexual abuse material and trafficking. Available at <https://endsexualexploitation.org/articles/a-look-into-onlyfans/>. Accessed in December 2021, June 2021.
- [40] Aytug Onan. Two-stage topic extraction model for bibliometric data analysis based on word embeddings and clustering. *IEEE Access*, 7:145614–145633, 2019.
- [41] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. pages 1532–1543, 2014.
- [42] Jipeng Qiang, Ping Chen, Tong Wang, and Xindong Wu. Topic modeling over short texts by incorporating word embeddings. September 2016.
- [43] Juan Ramos. Using tf-idf to determine word relevance in document queries. *Proceedings of the first instructional conference on machine learning*, 242:29–48, December 2003.
- [44] Michal Rosen-Zvi, Thomas Griffiths, Mark Steyvers, and Padhraic Smyth. The author-topic model for authors and documents. July 2012.

- [45] Hannah Ritchie Max Roser and Esteban Ortiz-Ospina. Internet. *Our World in Data*, 2015. <https://ourworldindata.org/internet>.
- [46] Max Roser, Hannah Ritchie, and Esteban Ortiz-Ospina. Internet. *Our World in Data*, 2015.
- [47] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing Management*, 24:513–523, January 1988.
- [48] Gerard Salton and Michael J McGill. Introduction to modern information retrieval. 124, 1983.
- [49] Kentaro Sasaki, Tomohiro Yoshikawa, and Takeshi Furuhashi. Online topic model for twitter considering dynamics of user interests and topic trends. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 1977–1985, October 2014.
- [50] Amit Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24:35–43, 2001.
- [51] Robert D. Steele. *Open source intelligence*, pages 147–165. Routledge, 1 edition, January 2007.
- [52] Pedro Szekely, Craig A. Knoblock, Jason Slepicka, Andrew Philpot, Amandeep Singh, Chengye Yin, Dipsy Kapoor, Prem Natarajan, Daniel Marcu, Kevin Knight, David Stallard, Subessware S. Karunamoorthy, Rajagopal Bojanapalli, Steven Minton, Brian Amanatullah, Todd Hughes, Mike Tamayo, David Flynt, Rachel Artiss, Shih Fu Chang, Tao Chen, Gerald Hiebel, and Lidia Ferreira. Building and using a knowledge graph to combat human trafficking. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9367:205–221, October 2015.
- [53] Amir Uteuov and Anna Kalyuzhnaya. Combined document embedding and hierarchical topic model for social media texts analysis. *Procedia Computer Science*, 136:293–303, January 2018.
- [54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017-December:5999–6009, June 2017.
- [55] Konstantin Vorontsov and Anna Potapenko. Additive regularization of topic models. *Machine Learning*, 101:303–323, October 2015.
- [56] Wikipedia. Regular expression.
- [57] Wikipedia. Topic model. Available at [https://en.wikipedia.org/wiki/Topic\\_model](https://en.wikipedia.org/wiki/Topic_model). Accessed in January 2022.
- [58] Pengtao Xie, Diyi Yang, and Eric P. Xing. Incorporating word correlation knowledge into topic modeling. *NAACL HLT 2015 - 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Proceedings of the Conference*, pages 725–734, January 2015.
- [59] Zhiheng Xu, Rong Lu, Liang Xiang, and Qing Yang. Discovering user interest on twitter with a modified author-topic model. *Proceedings - 2011 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2011*, 1:422–429, 2011.

- [60] Xiaohui Yan, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. A biterm topic model for short texts. *WWW 2013 - Proceedings of the 22nd International Conference on World Wide Web*, pages 1445–1455, 2013.
- [61] Wayne Xin Zhao, Jing Jiang, Jianshu Weng, Jing He, Ee Peng Lim, Hongfei Yan, and Xiaoming Li. Comparing twitter and traditional media using topic models. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6611 LNCS:338–349, 2011.
- [62] Radim Řehůřek and Petr Sojka. Software framework for topic modelling with large corpora. pages 45–50. ELRA, May 2010.