UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MASTER

Design of a Prototype of a Big Data Analysis System of Online Radicalism based on Semantic and Deep Learning technologies

Rodrigo Barbado Esteban
2018

TRABAJO DE FIN DE MASTER

Título:	Diseño de un Prototipo de un Sistema de Análisis Big Data de Radicalismo en Internet basado en tecnologías Semánticas y de Apendizaje Profundo
Título (inglés):	Design of a Prototype of a Big Data Analysis System of Online Radicalism based on Semantic and Deep Learning technologies
Autor:	Rodrigo Barbado Esteban
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	—
Vocal:	—
Secretario:	—
Suplente:	—

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO DE FIN DE MASTER

Design of a Prototype of a Big Data Analysis System of Online Radicalism based on Semantic and Deep Learning technologies

Junio 2018

Resumen

El auge en los últimos años del terrorismo islámico en Occidente ha obligado a tomar medidas preventivas con el fin de evitar o minimizar nuevas tragedias. Este auge viene acompañado a su vez de una nueva forma de promoción de las organizaciones terroristas, que utilizan Internet como medio de difusión de propaganda para conseguir nuevos adeptos en todo el mundo. Un claro ejemplo de este hecho se dio en el atentado de Barcelona de agosto de 2017, en el que se descubrió que terroristas que participaron en el atentado habían compartido previamente mensajes de odio en sus redes sociales.

Este trabajo forma parte del proyecto Trivalent (Terrorism pReventIon Via rAdicaLisation countEr-NarraTive), en el cual se trata de dar una solución preventiva al problema descrito estudiándose las narrativas utilizadas por organizaciones terroristas con el fin de desarrollar contra-narrativas eficaces para prevenir los procesos de radicalización.

El trabajo se divide en dos partes. La primera describe un sistema de monitorización de medios de Internet como webs de periódicos, redes sociales o revistas de propaganda yihadista presentes en Internet. Este sistema consta de tres partes fundamentalmente: un primer paso consistente en la ingesta de datos, después un proceso de análisis de dichos datos y enriquecimiento de los mismos de carácter semántico siguiendo pautas de Linked Data, y por último el almacenamiento de la información extraída acompañado de una capa de visualización. Este sistema además ofrece la posibilidad de anotar narrativas sobre los textos extraídos para poder realizar otras operaciones como por ejemplo deteccion automática de narrativas.

La segunda parte consiste en la creación de un sistema de clasificación de tweets según tengan un componente de radicalismo o no. Para el desarrollo del clasificador se han utilizado tecnologías de Procesamiento de Lenguaje Natural y de Aprendizaje Profundo, estando el algoritmo realizado con la librería Keras que actúa sobre TensorFlow.

Palabras clave: Radicalismo, Scraping, Linked Data, Procesamiento de Lenguaje Natural, Aprendizaje Profundo.

Abstract

The rise in recent years of Islamic terrorism in the West has forced states to take preventive measures in order to avoid or minimize new tragedies. This boom is accompanied in turn by a new form of promotion of terrorist organizations, which use the Internet as a means of spreading propaganda to get new followers around the world. A clear example of this fact occurred in the attack of Barcelona in August of 2017, in which it was discovered that terrorists who participated in the attack had previously shared hateful messages on their social networks.

This work is part of the Trivalent project (Terrorism pReventIon Via rAdicaLisation countEr-NarraTive), which aims to give a preventive solution to the problem described by studying the narratives used by terrorist organizations in order to develop effective counter-narratives to prevent the processes of radicalization.

This thesis is divided in two parts. The first describes a system for monitoring Internet media such as newspaper websites, social networks or jihadist propaganda magazines present on the Internet. This system consists of three parts fundamentally: a first step consisting in the data intake, then a process of analyzing said data and enriching it by semantic means following Linked Data guidelines, and finally the storage of the extracted information accompanied of a visualization layer. It also offers the possibility of annotating narratives about extracted texts in order to carry out other operations such as automatic detection of narratives.

The second part consists in the creation of a classification system of tweets according to whether they have a radicalism component or not. For the development of the classifier, Natural Language Processing and Deep Learning technologies have been used, and the algorithm has been developed with the Keras library acting over TensorFlow.

Keywords: Radicalism, Scraping, Linked Data, Natural Language Processing, Deep Learning.

Contents

List of Figures

CHAPTER 1

Introduction

This first chapter describes the context of this project stating the reasons of its development, the proposed goals which were accomplished and the structure of the rest of the document.

1.1 Context

The rise of terrorist attacks in the last years has heavily increased its impact in people's lives. Several European countries such as France, England, Belgium, Germany or Spain have suffered recent attacks in a small time interval and some reactive measures have been taken by governments such as upgrading terrorist alarm levels to their maximum values.

This wave of recent attacks in Europe, managed mainly from Daesh, started on the 7th of January of 2015, when two terrorists entered the Charlie Hebdo offices in Paris opening fire against everyone they encountered. The balance was a total of twelve deaths, and six more people died the following days while persecuting the authors of the massacre. Also in 2015, on the 13th of November 129 people died due to a new attack mainly focused in the Bataclan concert hall.

2016 was also a year when Europe suffered cruel jihadist attacks. On March 22nd, Brussels subway attack resulted in 32 deaths. Months ĺater, on the 14th of July, France was celebrating its National Day but a truck attack cost the life of 84 people in Niza along with hundreds of injured people. Another truck attack provoked the death of twelve people in a Christmas market in Berlin on the 19th of December.

During the first months of 2017, several attacks occured in Paris. On the 22nd of March, five people died on the Westminster bridge near the British Parliament. Another 14 people died days later, the 3rd of April in a subway attack in Saint Petersburg. Four days later, another truck cost three lives in Stockholm. On the 22nd of May, 22 people died in the Manchester Arena during a concert due to an explosion.

On the 17th of August of 2017 Spain suffered this wave of Daesh attacks, 24 people died and more than 150 more were injured due to a van attack at Las Ramblas in Barcelona. In this last attack, important findings from one of the attackers were collected from social networks. One viral finding was discovered from his Ask.fm account, a social network for asking personal questions to users. The message posted in this network by the member of the jihadist organization (when he was still under 18) was "Kill the unfaithful. Only leave alive muslims who follow the religion". In fact, two of the authors of the massacre were below 17 years old, belonging to the age interval between 15 and 19 in which 34.2% of radicalisation processes occur according to a study made by Real Instituto Elcano [13].

From this case we extract not only that there were signs of radicalisation of the attacker which could be used to detect it in an early stage, but also that hateful propaganda is spread over social networks and anyone can consume it, leading to more radicalisation cases. In order to face this problem, several initiatives are being promoted by governments and different organizations so as to detect the presence and spread of dangerous information over the Internet.

One of the initiatives dealing with this problem is the RD H2020 project Trivalent [9] (Terrorism pReventIon Via rAdicaLisation countEr-NarraTive), and this thesis has been developed as part of the participation of the Intelligent Systems Group (GSI-UPM) on it. This project is in line with the UN Security Council recommendations and the Commission "European Agenda on Security" 2015-2020, in order to contrast successfully violent extremism needing a more balanced response to terrorism combining repressive (protective) measures with preventive measures. This work is done in collaboration with actors of civil society and the communities of reference based on a firm commitment to respect fundamental rights and fighting discrimination.

A better understanding of factors constituting violent radicalisation in Europe is needed from a multidisciplinary analysis, investigating its root causes for developing the right countermeasures which range from early detection methodologies to strategies, ways and techniques of counter-narrative.

Additionally, it is necessary to acknowledge that violent radicalisation in jihadist extremism goes mainly through narratives that have specific contents, are addressed to specific audiences and spread in several ways being the Internet one of them.

1.2 Project goals

As it has been stated before, the global goal of this project is to study the presence of radicalism in the Internet and the direct effect it has on society. We have detected several forms in which radicalism appears on the Internet: first, there exist propaganda sources directly written by jihadist organizations; secondly, informative sources such as online newspapers; and lastly, social networks, in which the previous contents are spreaded along with personal opinions which are used to pursue radicalist objectives in many cases.

Having the previous statements in mind, the following goals are defined:

1. Data ingestion from Internet contents, captured directly from the Internet or previously downloaded, from different sources and in different formats.
   (a) Information present in social networks.
   (b) Information gathered from online newspapers.
   (c) Information gathered from jihadist sources.
2. Data enrichment through semantic analysis following linked data principles.
3. Data persistence for storing the resultant data in order to make it reusable in further research.

4. Development of a deep learning system capable of detecting radical content with recruitment aims.

5. Conclusion extraction related to the results of the project and the presence of online radicalisation.

1.3 Structure of this document

The remaining of this document is structured as follows:

Chapter 2 describes the technologies that have been used for the development of the project.

Chapter 3 shows the radicalist data monitoring system developed for the project capable of scraping, analyzing, enriching and storing it, offering also a visualization layer.

Chapter 4 describes neural network topologies focused on text classification.

Chapter 5 shows the methodology and experiments carried out for developing a radicalist tweet classification Deep Learning based model.

Chapter 6 states the conclusions of the project and future work lines.

CHAPTER 2

Enabling Technologies

This chapter explains the different technologies used for the development of the project. The core code of the project has been developed using Python, an interpreted high-level programming language, and most of the experiments have been carried out using IPython notebooks, which work as an interactive computational environment where code can be executed by steps known as cells. For machine learning tasks, it has been needed to use different libraries divided into data managing libraries, natural language processing libraries and machine learning libraries. The process of these experiments is first loading a dataset from an external source, extracting useful information from them and finally passing this information through the machine learning algorithms.

2.1 Data managing libraries

In order to treat data for carrying out experiments, two main Python libraries outstand.

2.1.1 Pandas

Pandas [39] is an open source BSD-licensed library, useful for working with data structures providing powerful tools for data analysis purposes. The most common data structure is the DataFrame, a two-dimensional labeled data structure with colums of potentially different types. This results really helpful for loading datasets into a Python program, containing the different fields as the DataFrame columns, being each item of the dataset a DataFrame row. Some of the most important capabilities of Pandas are listed below:

- Easy handling of missing values.
- Columns can be inserted and deleted from DataFrames and higher dimensional objects.
- Powerful group by functionality which allows us to perform split-apply-combine operations for both aggregating and transforming data.
- Slicing, indexing and subsampling of large datasets.
- Merging and joining datasets.

2.1.2 Numpy

Numpy [53] is the fundamental package for scientific computing with Python. The library was used due to its powerful N-dimensional array object and its linear algebra capabilities. Those data structures and transformations are useful for transforming textual information into numerical features understandable by machine learning algorithms, being carried out by natural language processing means.

2.2 Natural language processing

This project has used several Python natural language processing libraries in order to preprocess textual data and extract information from it.

2.2.1 NLTK

NLTK [19] provides a suite of text processing utilities such as tokenization, stemming, tagging or parsing. It is really useful in the first phase of the analysis tasks, as it allows us to

transform texts into sentences and words, reducing their dimensionality through stemming and obtaining additional information such as POS tags. All this information can be encoded into numerical attributes afterwards using Numpy.

2.2.2 Gensim

Gensim is an open-source Python library for topic modelling, characterised by providing scalable statistical semantics, analyzing plain-text documents for semantic structure and retrieving semantically similar documents.

In this project, Gensim has been used for generating vector representations of words called word embedding, reflecting the semantic domain into a vectorial space through a algorithm called Word2Vec [29] which has been implemented in this library. This process will be deeply explained on the following sections.

2.3 Machine Learning

There are several machine learning libraries available for Python, frequently updated by a huge community of developers. Those libraries are in charge of performing machine learning tasks such as classification , and take numerical data as inputs which, in this case, has been obtained using the previous described technologies.

2.3.1 Scikit-learn

Scikit-learn [44] is a simple and efficient tool for data mining and data analysis built on Numpy, Scipy and Matplotlib. It is open source, commercially usable through a BSD license. It offers tools for performing the following tasks:

- Classification: identifying to which category an input belongs to. Several algorithms can be found here, such as Support Vector Machines, Random Forest, Naive Bayes...
- Regression: predicting a continuous-valued attribute, being some well-known algorithms SVR or ridge regression.
- Clustering: automating grouping or similar objects into sets. For example, the k-Means algorithm.
- Dimensionality reduction: reducing the number of random variables to consider through techniques such as PCA or t-SNE.
- Model Selection: comparing, validating and choosing parameters and models. Grid search is one of the most popular techniques for performing this task, and Scikit-Learn

also offers modules such as cross validation or metrics with the goal of choosing the best model.

- Preprocessing: feature extraction and normalization.

2.3.2 TensorFlow

TensorFlow [14] is an open source library for dataflow programming. It is a symbolic math library commonly used for machine learning applications such as neural networks. Its data flow is based on graphs consisting of several nodes which represent mathematical operations and graph edges representing the multidimensional data arrays communicated between them. Those multidimensional arrays are known as tensors.

Tensorflow graphs are fed with data through placeholders, which are unassigned until the session runs. Additionaly, there are other elements which require special attention known as Variables, which are the values that change during the training process. Those Variables are the matrices involved in each layer of the neural network, and change due to the optimization procedures which occur during training.

2.3.3 Keras

Keras [6] is a high level neural networks API capable of running over a TensorFlow, CNTK or Theano backend. Its development goal was focusing on enabling a fast experimentation, as it serves as a wrapper of the TensorFlow functionalities, facilitating the creation of neural network architectures. Its main characteristics are:

- User friendliness, modularity and extensibility.
- Supports both convolutional networks and recurrent networks, which are extensely used in the machine learning field.
- seamlessly Runs on CPU and GPU.

2.4 Elasticsearch

Elasticsearch [30] is a distributed, RESTful search engine. It provides a multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. It is released as open source with an Apache License and can be used to search all kinds of documents also providing scalability capabilites.

2.5 Senpy

Senpy [48] is a generic framework for natural language processing services developed by the GSI group from UPM. It provides the possibility to easily use and interchange its services as they share a common API, and also simplifies the service development process. During service development, it includes the following functionalities:

- Parameter validation and error handling.
- Formatting, which can be made for example using JSON-LD, Turtle, text input...
- Semantic annotation in favour of Linked Data, based on the following specifications:
  - Marl, a vocabulary for designing and describing subjective opinion from the web or information systems.
  - Onyx, built on the same principles as Marl with the aim of annotating and describing emotions.
  - NIF 2.0, which defines a semantic format and API for improving interoperability among natural language processing services.
- A web user interface where users can interact with the different developments.
- A Python client which interacts with the service.

```
<http://microblog.com/User1/Post1#char=0,49>
  rdf:type nif:RDF5147String, nif:Context;
  nif:beginIndex "0";
  nif:endIndex "75";
  nif:isString "The example they used in their last paper
      ↪ was very clear, I really liked it";
  marl:hasOpinion :Opinion1.
:Opinion1
  rdf:type marl:Opinion;
  marl:describesObject "paper";
  marl:describesObjectPart "example";
  marl:describesFeature  "clarity";
  marl:polarityValue "0.8";
  marl:hasPolarity: marl:Positive;
  prov:wasGeneratedBy :Analysis1.

:Analysis1
  rdf:type marl:SentimentAnalysis;
  marl:maxPolarityValue "1";
  marl:minPolarityValue "-1";
  marl:algorithm "dictionary-based";
  prov:wasAssociatedWith http://www.gsi.dit.upm.es/.
```

Figure 2.1: Semantic output.

2.5.1 Framework

Senpy framework is divided into five different layers, which helps transferring modularity to its implementations reducing the cost of adding new features and algorithms.

These are the five layers:

- Analysis Layer: includes the core natural language processing process and the libraries to connect it to the other layers.
- Semantic Layer: deals with conceptual models and integration.
- Syntactic Layer: handles issues such as formatting, serialization and I/O validation.
- User interface: offers the possibility for users of interacting with services.
- Evaluation Layer: allows users to benchmark different algorithms.

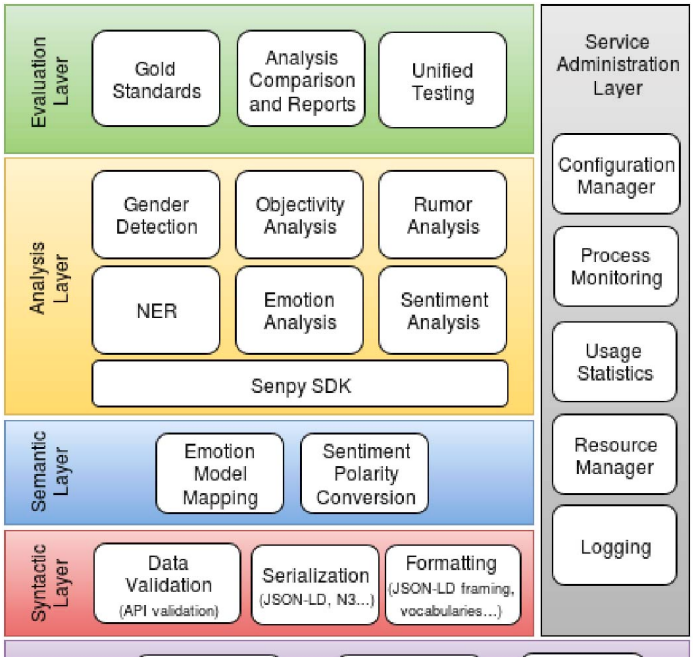


Figure 2.2: Senpy framework.

2.5.2 Architecture

The architecture consists of two main modules, which are Senpy core and Senpy plugins.

- Core: it provides the main functionalities of the platform, which are an HTTP server/-CLI interface, parameter extraction and validation, serialization of results along with

their publication as Linked Data. It offers two ways to access it: through a command line interface or a HTTP server. Both ways use an API aligned with NIF but using a JSON-LD representation and a JSON-schema by default. The API also defines the allowed parameters and extra parameters that each plugin declares when defined. Once the analysis has finished, results are modified before being returned to the user through transformers which map them according to the user's requirements. Finally, results are offered in the right format including metadata and URI identifiers for publishing it as Linked Data.

- Plugins: they represent analysis processes. They are defined by two elements: a definition file (written in JSON or YAML with .senpy extension) and the plugin code. The definition file contains information related to the plugin such as name, version, parameters needed and attributes of the plugin.

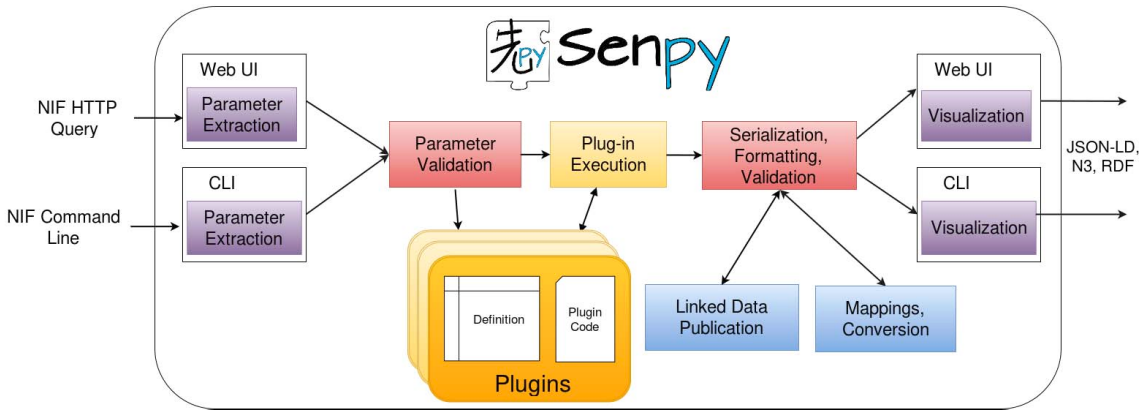


Figure 2.3: Senpy architecture.

2.6 GSI Crawler

GSI Crawler [3] is an innovative and useful framework which aims to extract information from web pages enriching them following semantic approaches. There are several available platforms included into GSI Crawler, which can be divided in three categories: social networks, online press and offline documents. Twitter and Facebook are available for the social category; several online newspapers are included as The New York Times, CNN or Al Jazeera; and finally jihadist propaganda magazines such as Dabiq or Rumiyah are examples of offline documents. GSI Crawler can be divided from a high level abstraction as follows:

- Data ingestion: this is the core function of GSI Crawler, consisting on extracting data according to certain queries. It works thanks to the use of web and PDF scrapers.

Additionally to the proper extraction of information, it is also kept in an structured way following semantic approaches fostering Linked Data capabilities.

- Data analysis: before its storage, data is enriched with several analysis modules available at the platform which can be concatenated forming pipelines. Those pipelines are composed of enrichment modules applied to the obtained data with the aim of augmenting its information, being those modules Senpy Plugins, which offer a common environment making them interoperable.
- Data storage: after data acquisition and enrichment, the storage process is carried out. At this moment, both ElasticSearch and Fuseki [50] are available for fulfilling this task.

2.6.1 Architecture

The following figure describes the architecture from a modular point of view, being each of the modules described below. GSI Crawler is formed by two main modules: a task server which manages all the information flow and a Web Application which acts as the front-end.

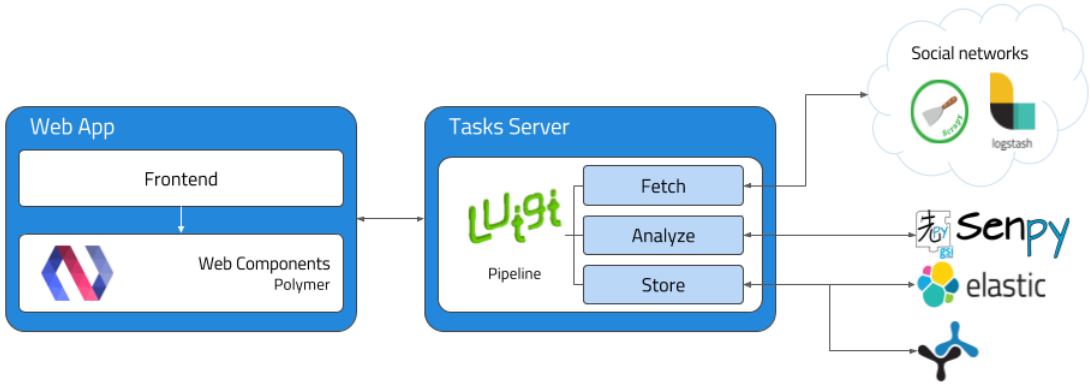


Figure 2.4: GSI Crawler architecture.

2.6.1.1 Tasks Server

The tasks server is responsible of managing all the information to be processed over the system from its ingest to its storage passing through an analysis process, being all the flow

carried out in a pipeline form. The data ingestion can be both periodical or static, the analysis process is done using Senpy plugins and the the results are stored in ElasticSearch to be displayed in the client application and in Fuseki for performing semantic queries over them. The orchestration is done through the Luigi framework [27], which allows us to build a sequence of tasks in the desired pipeline form..

This tasks server is periodically activated by an administrator of processes called cron, whose aim is to obtain more information everyday. That way, any user can visualize data any time with the certainty that there will be new stored data in the system.

All the pipelines share the same structure, as represented in the figure below.

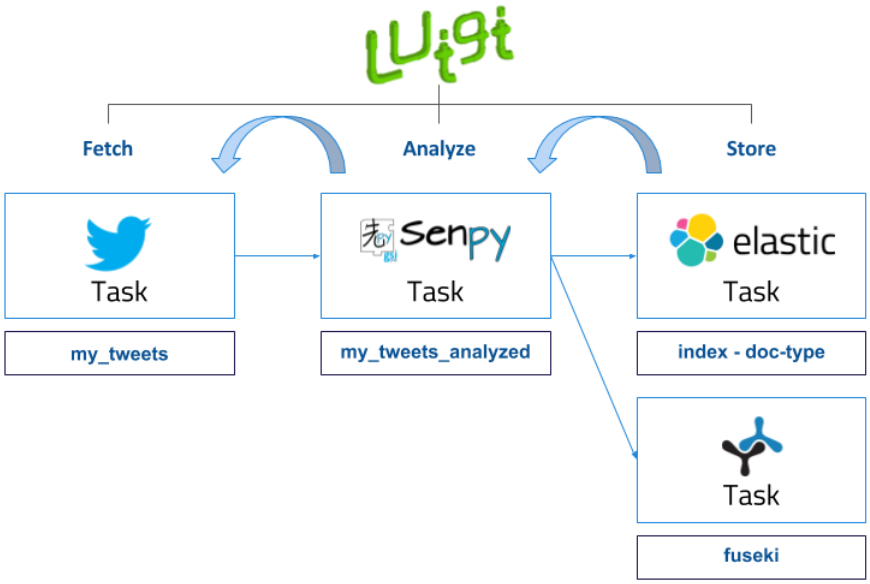


Figure 2.5: GSI Crawler pipeline structure.

As represented above, pipelines architecture is divided into three main steps: Fetch, Analyze and Store. Those are their main goals:

- Fetch refers to the process of obtaining the content which is desired to be analyzed from its resource identifier. Most of the times, this task involves webpage parsing, recognizing valuable information contained inside HTML tags and building new JSON files with the selected data. This process is commonly known as web scraping. In order to facilitate this filtering process, there exist multiple extensions or libraries that offer a well-formed structure to carry out this task in a more comfortable way.

  Inside the Tasks Server, the Scrapy [42] Python library has been imported in order to facilitate the data mining process, which is used along with several APIs.

This Fetch step also includes data ingestion from offline data sources of documents which had been previously downloaded. For example, GSI Crawler is able to get PDF files as input and extract their inner information. The process of obtaining data form documents is commonly known as Document scraping, and in the particular case of PDF files, PDF scraping.

The resulting data extracted from this step generates a JSON which can be used by the following task in the pipeline. Additionally, this JSON file contains information organised according to Linked Data principles in order to provide semantic analysis capabilities.

- Analyze task is responsible of taking the input JSON generated by the previous task, parsing it and performing the desired analysis using Senpy plugins. Senpy plugins are services based on HTTP calls which allow to obtain analyzed results for the text attached in the request. Once the task has collected the analysis result, it generates another JSON containing the original JSON along with the added information.

  Additionally, Senpy plugins can be concatenated making use of the pipeline structure of GSI Crawler and the easy interoperability between them as they are designed as independent modules which take pre-known inputs and output well-structured results.

- Store process consists on storing the JSON resulting from the last analysis task inside ElasticSearch, which is a distributed, RESTful search and analytics engine capable of solving a growing number of use cases. To carry out the saving process, it's necessary to provide two arguments: the index, which represents the elastic index where the information will be saved, and the doc type, which enables to categorize information.

  Additionally to ElasticSearch, the results can also be stored in Fuseki, a semantic oriented database which allow to perform queries based on Linked Data principles known as SPARQL [46] queries.

2.6.1.2 Web App

GSI Crawler framework uses a web page based on Polymer web components to interact with all the functionalities offered by the tool. Those Polymer Web Components are simply independent submodules that can be grouped each other to build the general dashboard interface.

2.7 Linked Data

Linked data is a way of publishing structured data in order to make it interlinkable and to enable a more useful way of querying it through semantic queries. It is based on standard web technologies such as HTTP, RDF [36] and URIs used to share information in a way to be automatically read by computers.

Linked data follows four principles stated by Tim Berners-Lee, as stated in his "Linked Data" note of 2006 [20]:

1. Use URIs to name (identify) things.
2. Use HTTP URIs so that things can be looked up.
3. Provide useful information about what a name identies when it's looked up, using open standards such as RDF or SPARQL.
4. Refer to other things using their HTTP URI-based names when publishing data on the Web.

He additionally restated the linked data principles in 2009 as three "extremely simple" rules [20]:

- All kinds of conceptual things, they have names new that start with HTTP.
- If I take one of these HTTP Names and I look it up... I will get back some data in a standard format which is kind of useful data that somebody might like to know about that thing, about that event.
- When I get back that information it's not just got somebody's height and weight and when they were born, it's got relationships. And when it has relationships, whenever it expresses a relationship then the other thing that it's related to is given one of those names that starts with HTTP.

The way of representing linked data is based on RDF (Resource Description Framework), in which anything can be a resource or entity (from a document to a physical thing). The way of expressing information is done using triples which have the following parts:

- *Subject*: a resource identified by a URI.
- *Predicate*: a URI identified relationship.
- *Object*: a resource or literal to which the subject is related.

The RDF graph is formed by sets of triples, and there exist several syntaxes in order to describe them:

- **RDF/XML**
- **Turtle, N3** [32] (Turtle family of RDF languages)
- **JSON-LD** [25] (JSON for Linked Data)
- **RDFa** [15] (embedding RDF into HTML)

On top of RDF, there are additional capabilities that can be added. For example, RDF Schema (RDF-S) supports the definition of vocabularies, which enrich the information representation in an easier way. Some popular RDF-S vocabularies are the following:

- **FOAF** [22] (Friend of a Friend), highly used in social networks.
- **SKOS** [41] for thesauri and taxonomies.
- **Schema.org**, aimed at web pages for search engines.
- **Dublin Core**, metadata of pages (creator, author...)

In order to query and manipulate RDF data, there exists an W3C standardised language called SPARQL, which stands from SPARQL Protocol and RDF Query Language. It has the following family of specifications:

- *SPARQL Query* - Graph patterns, such as SELECT query (returns variable binding), ASK query (yes/no question), CONSTRUCT query (a new RDF graph is constructed from the query result)
- *SPARQL Federated Query*: delegates subqueries to SPARQL endpoints.
- *SPARQL Update*: Insert, Delete and Update RDF triples.
- *SPARQL Protocol*: Graph operations over HTML.

2.7.1 Ontologies used in this project

For the development of this project, the following ontologies have been mainly used:

- *NIF* [34]: it stands from Natural Language Processing Interchange Format, and is an RDF/OWL-based format which aims to achieve interoperability between NLP tools, language resources and annotations. It contains seven core URIs: String, RFD5147String, Context, isString, referenceContext, beginIndex and endIndex). It has been used for annotating entities appearing on some of the analyzed texts of the project.

- *PROV-O* [18]: it expresses the PROV Data Model, providing a set of classes, properties and restrictions that can be used to represent and interchange provenance information generated in different systems and under different contexts. In this project, it has been used to describe how the information was generated from the original data sources.

- *Schema* [31]: it can be used with many different encodings, including RDFa, Microdata and JSON-LD. These vocabularies cover entities, relationships between entities and actions, and can easily be extended through a well-documented extension model. It has been used for defining the extracted entities from the analyzed texts.

2.7.2 RDFLib

RDFLib [37] is a Python package for dealing with RDF, which contains the following utilities:

- Parsers and serializers for RDF/XML, N3, Ntriples, N-Quads, Turtle, TriX, RDFa and Microdata.

- A Graph interface which can be backed by several Store implementations.

- Store implementations for in memory and persistent storage using the Berkeley Database.

- A SPARQL 1.1 implementation.

The primary interface exposed by RDFLib for working with RDF is known as Graph, and the main operation which can be used for constructing it is called add(), which takes three parameters: subject, predicate and object. It can additionally serialize existing graphs defined in multiple formats and convert them into another one.

2.8 Neural Networks fundamentals

This section describes the basics of Neural Networks [49], from their simplest architecture to a multilayer perceptron used in general purpose applications, describing also their learning and optimization processes.

Neural networks behave like function approachers, and in this work will be used for solving supervised learning problems in which we have some input information and a desired output for each instance. The goal is creating a function which, using input information, outputs values which are the most similar to the real ones.

2.8.1 Perceptron

The simplest model is the perceptron, which consists of a simple neuron and was developed by Warren McCulloch and Walter Pitts based on previous work done by Ramón y Cajal. This neuron model was lately extended by Frank Rosenblatt [47], and its score was obtained as a weighted linear combination of the inputs. If the score was superior to a certain threshold, the output was a 1, and a 0 if not.

Its mathematical formulation is:

$$f(x) = \begin{cases} 1 & \text{if } \mathrm{w}_1 x_1 + ... + w_i x_i + ... + w_N x_N > threshold \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Those w parameters are the components of the weight matrix which multiplies the input, and there is also an additional bias component added to the resultant product as shown in Figure 2.6.

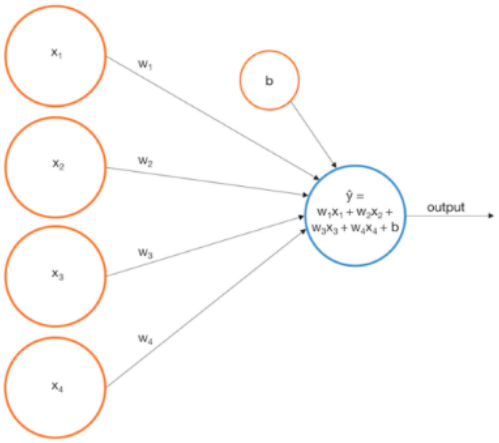


Figure 2.6: Single Layer Perceptron [5]

This model has two main problems: first, the model outputs real numbers which are not inside an specific range and cannot be interpreted as probabilities; and second, it can only learn linear relationships between variables and the target as the operations carried out are simple linear combinations.

For fixing the mentioned problems, a non-linear function is added at the end of the perceptron. Initially, this non-linearity was achieved by applying the sigmoid function of the previous output. The reason of using a sigmoid is because of its range domain between 0 and 1 and also is frequently used to model the probability of e binary event. The sigmoid function is described as follows (Figure 2.7):

$$sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

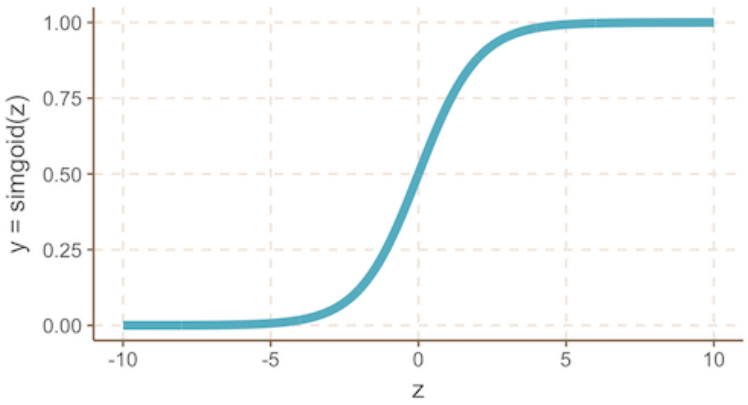


Figure 2.7: Sigmoid function [5]

Introducing this function, the new equations are the following:

$$z = w \cdot x = w_1 x_1 + ... + w_i x_i + w_N x_N; \widehat{y} = sigmoid(z) \tag{2.3}$$

This model is equivalent to the logistic regression (Figure 2.8), and now the output is bounded between 0 and 1 which can be interpreted as a probability. Thus, the decision rule is updated:

$$f(x) = \begin{cases} 1 & \text{if } \widehat{y} > 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

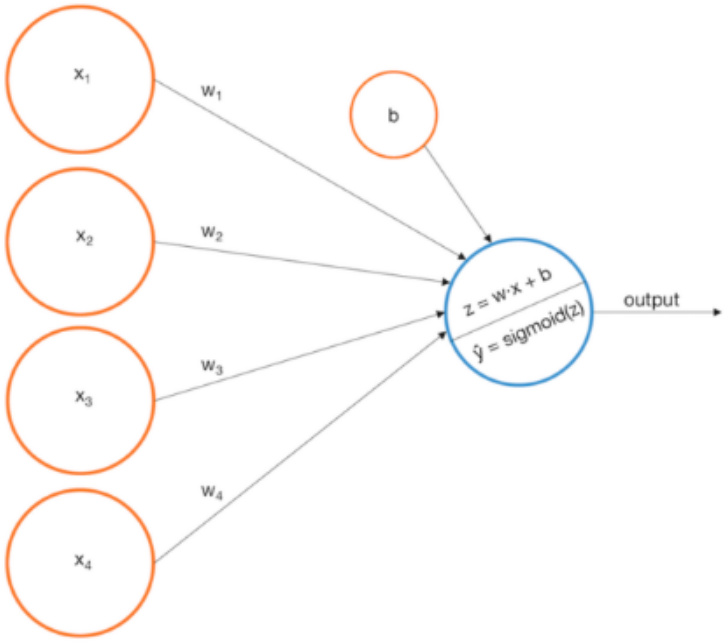


Figure 2.8: Perceptron with sigmoid activation [5]

However, there are still some issues regarding this improved model. The output has a monotonic relationship with each variable and the model doesn't use variable interaction for computing the output.

2.8.2 Multi-Layer Perceptron

Making use of the individual neuron or perceptron units which were described previously, it is possible to create more complex architectures which solve the problems described. It is possible to combine lots of neurons arranging them in layers which take as input the output of their previous one.

The layer terminology used for this architectures with more than one perceptron are an input layer composed of the data which enters the network, hidden layers consisting on several perceptrons which don't interact between them but with the previous and following layer, and an output layer. A graphic representation of a simple multi-layer perceptron is shown in Figure 2.9.

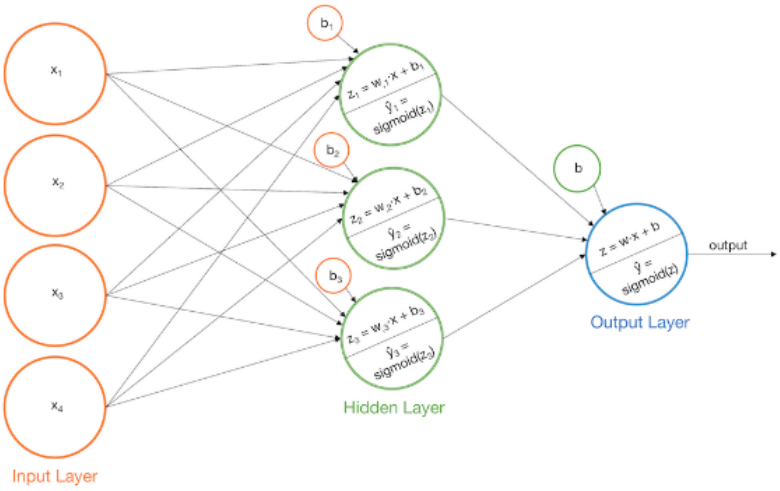


Figure 2.9: Simple Multi-Layer Perceptron [5]

Apart from Sigmoid function, another activations can be applied to neural networks. Two of the most used ones are *tanh* (hyperbolic tangent, Figure 2.10) and *ReLU* (rectified linear unit, Figure 2.11).

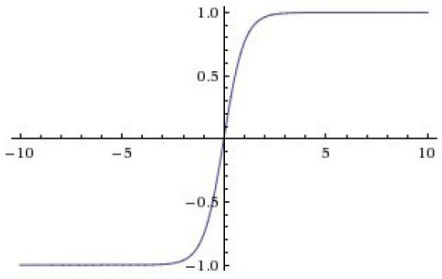


Figure 2.10: Tanh activation.

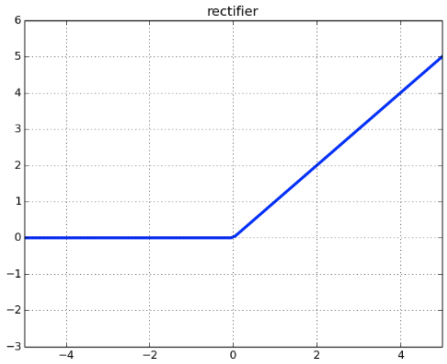


Figure 2.11: ReLU activation.

2.8.3 Neural Network Training

Once neural networks have been defined, this section will show how they are capable of learning which output corresponds to each input so as to perform a task in the most optimal way. This process of learning the optimal weights of each neuron in the network is known as training, and the measure of correctness of a neural network output is known as cost [28]. The cost of a prediction intuitively states how far the prediction was from the real value it should have, and from this metric we obtain the cost function. The goal of the training process will be to obtain the lowest value of this cost function as that would mean that the errors are minimised.

2.8.3.1 Cost function

The cost function should be ideally convex, as that would mean that its minimum would be a global minimum. However, in reality those functions are not truly convex and so their optimization process gets harder. Additionally, as this cost function is parametrised into the weights vector space, the increase of weights in our network architectures also increases complexity in the training process and so it is more difficult to find the optimal solution.

Another issue to take into account while optimising our network is known as overfitting, which is a common problem in all machine learning models and means that the model adapts perfectly to the training examples but isn't capable of generalising the problem solution. That way, new examples are hardly correct as they have not been seen by the network before. For managing this problem, there exists a regularization term which is added to the cost function.

Once this cost function has been defined, typically denoted as J(W), the goal is to obtain the weights that minimize it and according to calculus the right way to carry it out would be obtaining its derivative and making it 0 to find where the significant points (minimum or maximum) are. However, that cannot be done easily due to some issues:

- The cost function cannot be easily expressed and its derivative would not be trivial.
- The dimensionality could be huge as it depends on the weights present at the network.
- Lots of minimum and maximum points would be found as the function is not strictly convex.

For those reasons, this problem's solution comes with iterative optimization algorithms which go finding the best solution step by step, using the available training data for it. The simplest method is called gradient descent, in which weights are initialized randomly and we start in any point of the function. The derivative is taken at that point, and it indicates the

direction where we need to move in order to obtain the minimum. Repeating this process, we would find the minimum point. This is translated into an update rule which is defined as follows:

$$W := W - \alpha \frac{\partial J}{\partial W} \tag{2.5}$$

This means that weights update is done by substracting the partial derivative of the cost function with respect to the weights scaled by a learning rate represented as alpha. The learning rate controls how big weight corrections are, and suppose a trade-off between speed of the training process and optimality of the reached solution. If the learning rate is too high, the variations are too big and the process could not converge. On the other hand, small learning rates could make the process slow or it could even get stuck at a local minimum.

2.8.3.2 Backpropagation

The output of a neural network is a composite function of its weights, inputs and activation functions, and for that reason the derivative computation is not as easy as for other models. Supposing we have a two-layer neural network, its mathematical formulation would be:

$$output = act(w3 * hidden2)$$

$$hidden2 = act(w2 * hidden1)$$

$$hidden1 = act(w1 * input)$$

Which can be expressed as:

$$output = act(w3 * (act(w2 * (act(w1 * input))) \tag{2.6}$$

When taking the derivative of the output with respect to any weight, the chain rule has to be applied. In this example, the chosen weight is w1:

$$\frac{\partial}{\partial w1} output = \frac{\partial}{\partial hidden2} output * \frac{\partial}{\partial hidden1} hidden2 * \frac{\partial}{\partial w1} hidden1 \tag{2.7}$$

Taking into account the cost function, another dependency is added and the error turns into a function of the output, and so function also of inputs, weights and activation functions. The structure would be as shown in Figure 2.12 :

And the resultant error derivatives [33]:

$$\frac{\partial error}{\partial w1} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial w1}$$

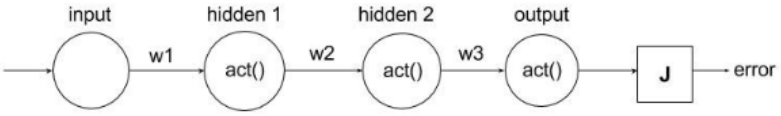


Figure 2.12: Neural Network with cost function.

(2.8)

At that point, it would be possible to apply the gradient Gradient Descent [21] iterative algorithm. For training the entire neural network, this has to be done for each wight present at each layer. The whole process is done following several steps:

1. Obtaining the neural network and training data.
2. Initializing the weights of the neural networks to random values, typically using normal distributions.
3. Performing one feed-forward using training data, that is to say, passing some examples of the training data and computing the cost metric of their output.
4. Doing backpropagation in order to get the derivatives of every weight in the network.
5. Update the weights using the error derivatives and performing gradient descent, which will update each weight by multiplying the negative scalar of the error derivative times the learning rate.
6. If convergence has been reached or a maximum number of iterations has been carried out, the training process ends. If not, the process has to be repeated from step 3 onwards.

2.8.4 Word Embeddings

Natural Language Processing is a concrete field inside all application fields treated by machine learning and thus it has some specific characteristics which are not used in other fields such as image or audio processing. One of the most remarkable differentiator between all fields are their input format: when doing NLP, inputs are numerical vectors representing the content of some text; but when doing for example image processing the inputs are matrices representing the pixel value for each position on the image. The problem of feeding text as vectors into a Neural Network has been studied for years, and the most common solution is creating the so-called Word Embeddings [52], which are mappings from words to vectors using a kind of "dictionary".

The simplest way to carry out this task can be done by one-hot encoding each word appearing on the text. For example, if we have the sentence "This is an example", we could have a dictionary which could be like this: ["This","is","an","example"]. For labeling words, we would use a four dimensional vector of zeros and a one in the position of the index belonging to the word in the dictionary. For example, "This" would be represented as [1,0,0,0] and "example" as [0,0,0,1]. This method is the simplest one for creating Word Embeddings, but has two main issues:

- The length of each word vector has as size the dimensionality of the entire vocabulary. So, when working with huge datasets, each input vector would have a extremely high dimension and so the training process would be harder.

- The inherital semantic relationships present in language are not kept by this model, as the distance between each pair of words is the same for the entire vocabulary domain. It would be desirable for example that words with similar meaning would appear together in the vector space.

Apart from the trivial case, we can classify Word Embeddings into two different categories: frequency based embeddings and prediction based embeddings. The first ones have been used in traditional machine learning algorithms when solving NLP tasks, being count vectors and TF-IDF [16] vectors their most popular techniques. However, more advanced methods following the prediction approach have been used by learning representations using Neural Networks which serve as input for other Neural Networks. The most popular technique for achieving prediction based vectors is called Word2Vec, which is a combination of two other techniques: CBOW and Skip-gram [45].

2.8.4.1 CBOW

CBOW [45], or Continuous Bag of Words, consists on predicting the probability of a word given a context formed by a group of words (Figure 2.13). The input layer consists of the one-hot encoded input context words x1,..., xC, being C the context window size and V the total vocabulary size. The hidden layer, denoted as h, is an N-dimensional vector, and the output layer is the output word y of the training example also one-hot encoded. The input vectors are connected to the hidden layer through W, a VxN weight matrix; and the hidden layer is connected to the output layer through W', of dimensions VxN. The hidden layer size represents the dimensionality of the resulting word vectors, and the weight between the hidden and output layer represents the word vector of each word.

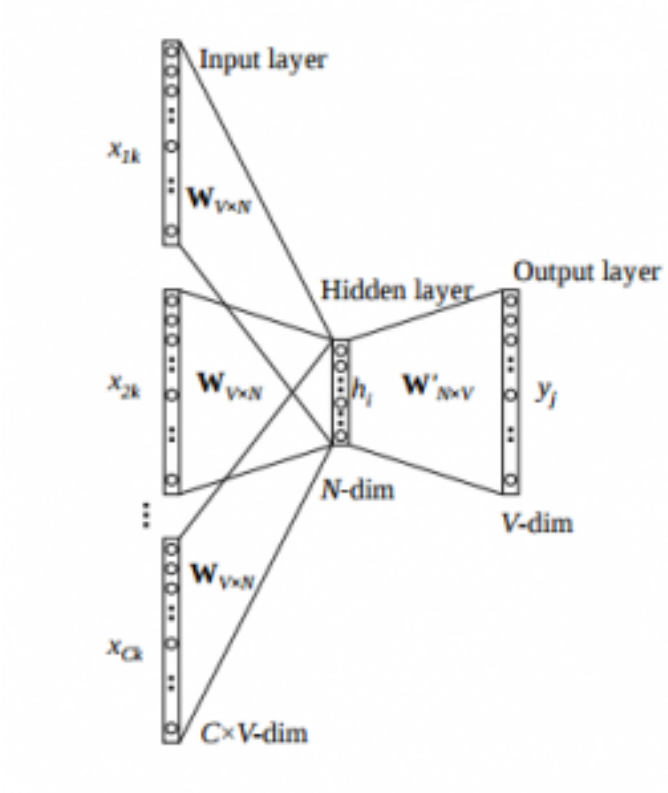


Figure 2.13: CBOW model [45]

2.8.4.2 Skip-gram

The Skip-gram model inverts the topology of the CBOW model [45], being its aim predicting the context words given a word. In this case, the weights between the input and the hidden layer represent the word vector representation (Figure 2.14).

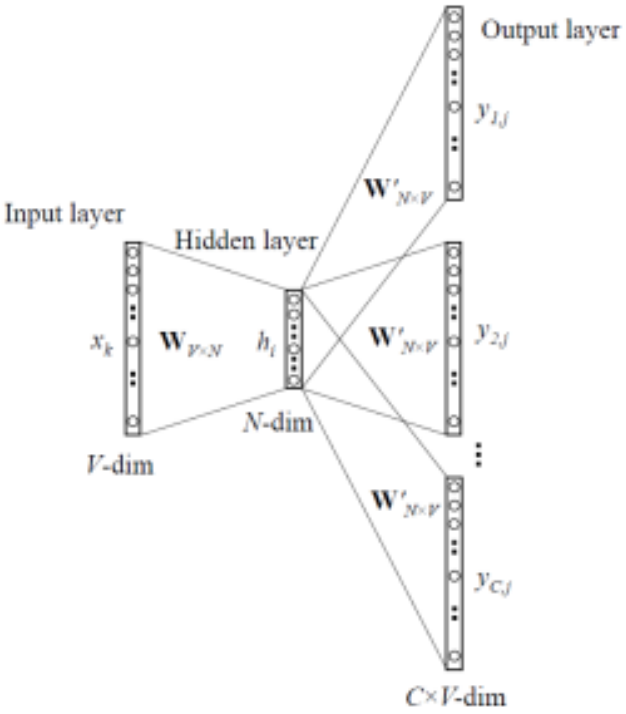


Figure 2.14: Skip-gram model [45]

The final word embeddings can be obtained by choosing one of the mentioned methods or combining them (for example by simple averaging). As a result, it is possible to visualize certain relationships by plotting the resultant vectors to lower spaces. Some of the kept relationships could be Male-Female, Verb tenses or Country-Capital (Figure 2.15). This supposes a huge advance with respect to using naive representations which don't add any information to the process.

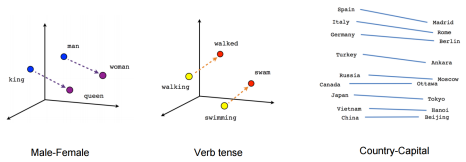


Figure 2.15: Word2Vec relationships [45]

2.8.5 Recurrent Neural Networks

Traditional Neural Network architectures as the Multi-layer perceptron are able of solving a great amount of general problems, but there are some others in which it's unclear to explain their effectiveness. For example, predicting the next event in a sequence would be difficult for Neural Networks in which the whole input is fed at the same time. For solving this problem, another type of architectures are used known as Recurrent Neural Networks [40]. Those networks have loops inside for persisting information. Their basic architecture is shown in Figure 2.16.

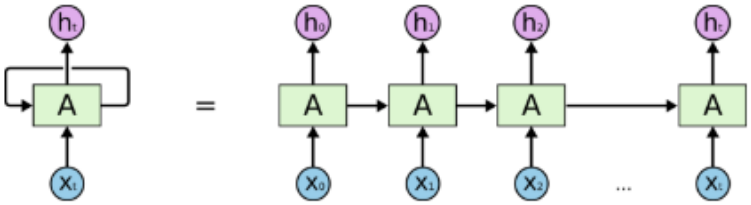


Figure 2.16: RNNs [12]

This architecture can be seen as having a single Neural Network replicated at different time steps, being each time step fed by their corresponding input and an output coming from its previous time step. That way, they look like sequences and perform well for those kind of problems. Some known tasks which can be modelled through this architecture are speech recognition, language modelling, neural translation or image captioning.

However, Recurrent Neural Networks or RNNs have some issues to be solved in their vanilla version. Although they can perform well when time dependencies occur in the short term, they have problems when those dependencies extend to much prior time steps. As this distance increases, their performance is reduced. This problem is technically issued as vanishing gradient problem. In order to solve it, two modifications stand out: LSTM [51] and GRU [23] networks.

2.8.5.1 LSTM Networks

LSTM [51] Networks stand for Long Short Term Memory Networks, which are an improvement of standard RNNs capable of dealing with long-term dependencies. Their goal is remembering past information, and that supposes a change in their architecture. In the case of standard RNNs, their repeating module have a simple structure such a hyperbolic tangent layer. However, the repeating module in LSTMs have four interacting layers. The most important part of the diagram below is the cell state, represented by the horizontal line on the top part. This line represents the flow of information along the network, and

this structure is also capable of adding or removing information to it through the so-called gates. Figure 2.17 shows an LSTM cell architecture.

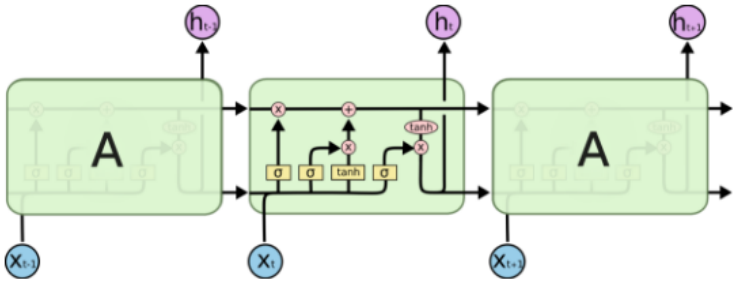
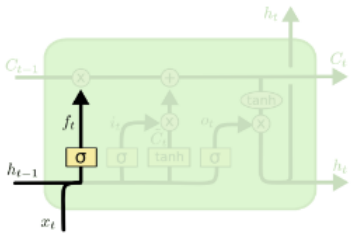


Figure 2.17: LSTM architecture [12]

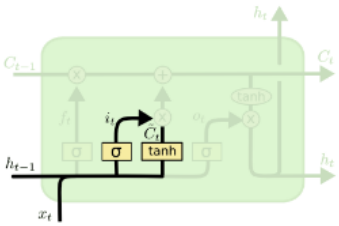
The first component of the module is known as forget gate layer (Figure 2.18), taking the hidden state of the previous step and the input at the current time step, and outputting a number between 0 and 1 for each number at the cell state. If the number is 1 all the information is kept whereas a 0 means to forget completely the information.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 2.18: LSTM forget gate [12]

Next, it is decided what new information coming from the previous step is stored in the cell state. It has first a input gate layer composed of a sigmoid layer to decide which values are updated, and also a tanh layer for creating a vector of new candidate values which could be added to the state. This can be shown in Figure 2.19.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 2.19: Deciding to store new information [12]

For updating the old state cell, it has to be multiplied by the forget output. Also, the

product of the two previous seen component has to be added. This process is depicted in Figure 2.20.

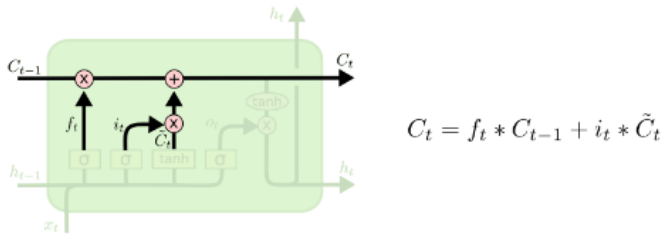


Figure 2.20: State update [12]

The last step is deciding the output of the module, filtering the cell state through several operations. First, a sigmoid function to decide the parts of the cell state which are going to be output, then a hyperbolic tangent to bound its values between -1 and 1 and finally multiplying both results (Figure 2.21).

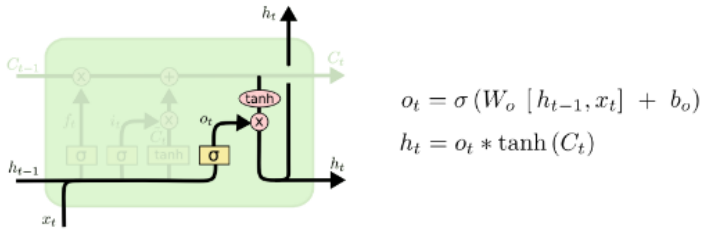


Figure 2.21: LSTM output [12]

2.8.5.2 GRU networks

Gated Recurrent Unit[23] networks also suppose an improvement over the standard RNN networks and is based on an update gate and a reset gate, which decide which information to pass. It rely on the same idea as LSTMs, but are different in their implementation.

First, the input is multiplied by a weight matrix and added to the product of the previous cell state and its weight matrix. After that product, the result is passed through a sigmoid function. This is the so-called update gate (Figure 2.22), which helps the model determining how much information is passed to future steps. That allows the network to copy the necessary information from the past so as not to make it disappear.

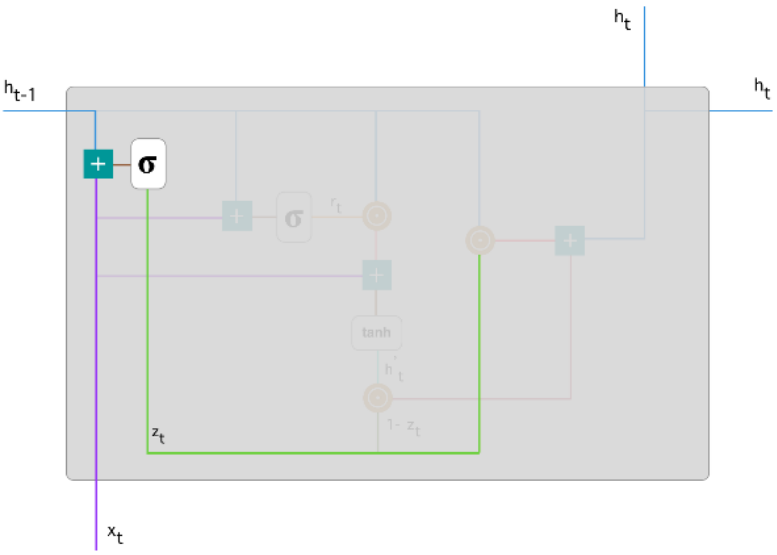


Figure 2.22: GRU update gate [11]

After the update gate comes the reset gate (Figure 2.23), which is used to decide how much of the past information is going to be forgotten. The method is similar to the update gate, but now the weights are different.

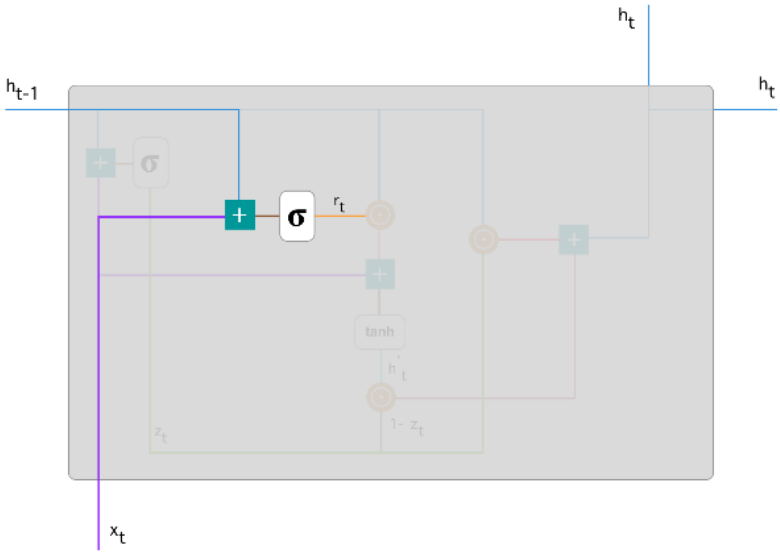


Figure 2.23: GRU reset gate [11]

Now, a new element called current memory content is added, which uses the reset gate to store the relevant information from the past. It first multiplies the previous state by its weights and the same for the input. Then, it makes the element-wise product between the reset gate and the previous state, sums it to the input and applies a hyperbolic tangent activation. This process is shown in Figure 2.24.

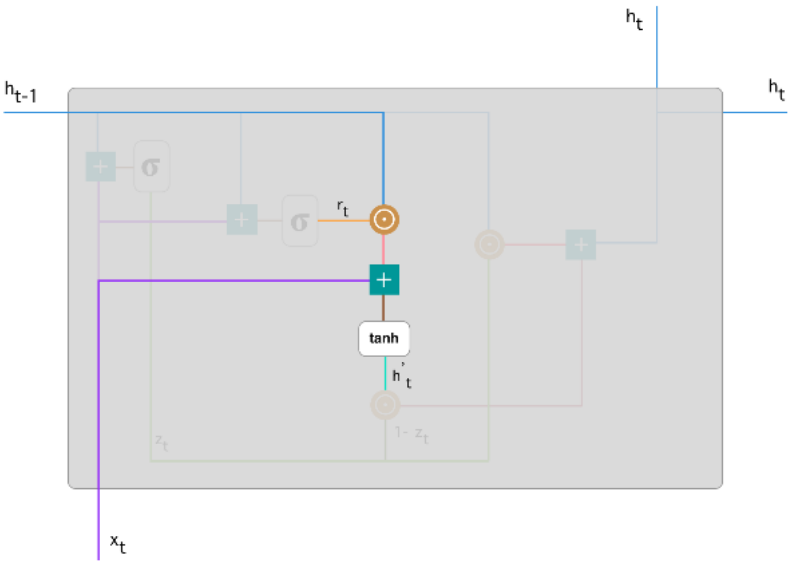


Figure 2.24: GRU current memory [11]

Finally, the new cell state is calculated in order to pass it to the following step. This steps selects the final information from the current memory content by applying the product of the update gate and the previous state, the product of the current memory content and the negative of the update gate plus one, and adding both results. Figure 2.25 shows this process.

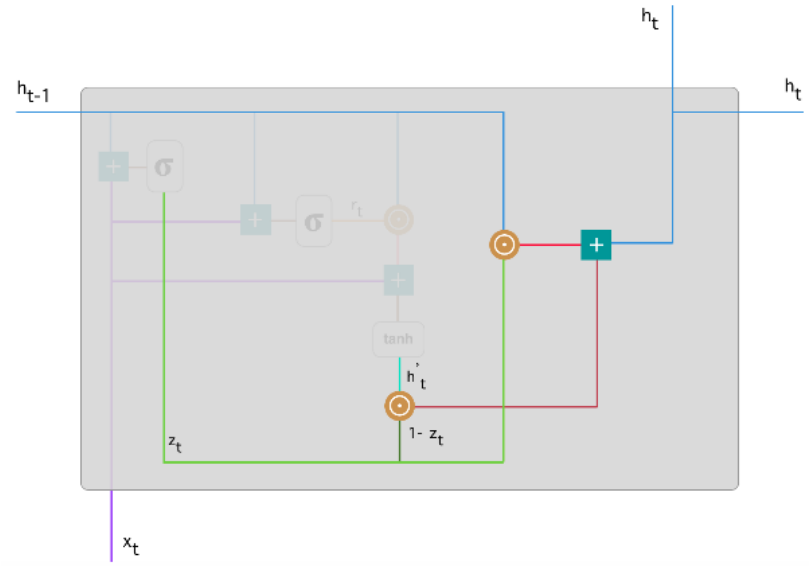


Figure 2.25: GRU final memory [11]

2.8.5.3 Attention over LSTM Networks

Attention mechanisms [55] in Neural Networks have been applied to NLP tasks in the last two years although it had been previously used in image recognition tasks. It is based on human visual attention, which focuses on a certain region of an image with high resolution while perceiving the rest in lower resolution and adjusts the focal point over time.

Translating the analogy to the NLP field, not all words have the same importance for detecting the intention of a message, so to understand what idea is transmitted humans have to focus on certain parts. Adding an Attention Layer over the LSTM network can provide this capability, and interesting improvements have been achieved in tasks such as sentiment analysis or neural machine translation.

There are several implementations of attention, but all of them have in common the reduction of information to create a context or summary vector. This is done using the following expressions:

$$u_{it} = tanh(W_w h_{it} + b_w) \tag{2.9}$$

$$\alpha_{it} = \frac{exp(u_{it}^T u_w)}{\sum_t exp(u_{it}^T u_w)} \tag{2.10}$$

$$s_i = \sum_t \alpha_{it} h_{it} \tag{2.11}$$

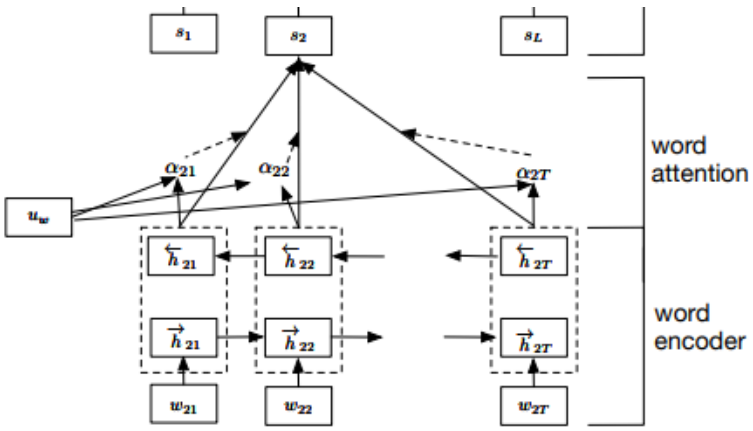


Figure 2.26: Attention over LSTM [56]

2.8.6 Convolutional Neural Networks

Convolutional Neural Networks [35] or CNNs have traditionally been used for image processing tasks, as its main mechanism is passing certain filters through a matrix in order to

detect specific shapes or patters which help in problems such as classification.

In the NLP field, instead of having a matrix compound of the pixel values of an image, we create a matrix formed by the word embeddings of a sentence. As a result, this matrix has as shape the number of words within a sentence multiplied by the word embedding size.

The goal of training a CNN is learning the values of the filters which are going to do the convolution operations. Additionally, pooling layers are introduced in this architecture for subsampling the outputs of each layer. The following image shows an example of a CNN architecture.

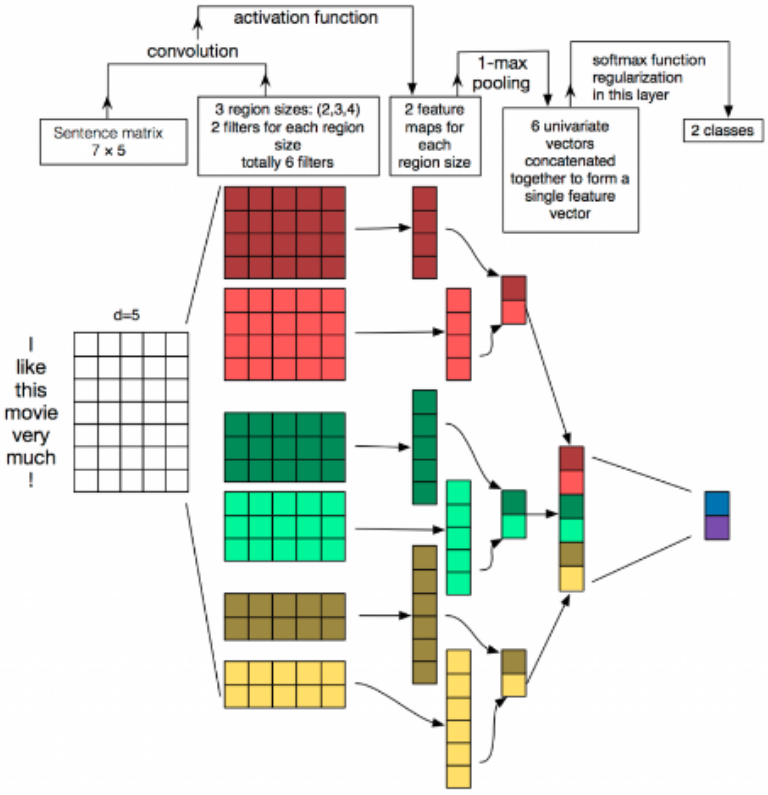


Figure 2.27: CNN architecture [10]

CHAPTER 3

Radicalist data monitoring

This chapter presents the monitoring tool built in this project, which is able to gather data from different sources, analyzing and enriching it, persisting it in no-SQL and semantic databases and visualizing it.

As stated in the project goals, a critical part for understanding the process of radicalization is gathering and analyzing data, and the Internet has become the greatest data repository. This chapter describes the used data sources and why they were chosen, the analysis performed over the gathered data for enriching it, the storage process and the visualization layer for understanding all the extracted information at a glance.

All the process relies over one of the described enabling technologies called GSI Crawler. This tool enables to create the workflow depicted in Figure 3.1, which starts with automated information gathering, continues with analysis and enrichment and ends with storage, allowing us also to build a visualization layer on top of it. Additionally, GSI Crawler is interoperable with Senpy, and so the analysis tasks have been developed as Senpy plugins.

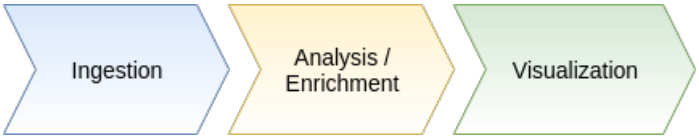


Figure 3.1: Definition of workflow.

There are three main sources of information in which jihadism is present over the Internet, and the following were selected within each category:

- **Online newspapers**: all news related with terrorism and middle East conflicts are closely followed by international press. The easiest way to capture this information is obtaining it through online newspapers. It is possible to search for news within desired categories, so we can use terms as "Isis" for only obtaining our target information. Two international online newspapers were chosen, The New York Times and CNN News, along with Al Jazeera, specialized on Middle East news.

- **Social networks**: the extended use of social networks over last years have made them become another news source and also a place where radicals try to spread their message in order to make other people join their organizations. We have included Twitter as it fosters written communication and community engaging.

- **Jihadist Propaganda**: terrorist organizations such as ISIS and Al-Qaeda have put effort on spreading their doctrines and beliefs over written media sources. In this case, we have chosen the two ISIS magazines which are distributed through the Internet, which are named Dabiq and Rumiyah. They are distributed in PDF format.

The following section will show the Linked Data Modelling of the designed solution. Afterwards, the next sections of this chapter show all the mentioned processes in detail, describing also their linkage with GSI Crawler and Senpy platforms.

3.1 Linked Data Modelling

One of the goals of this project is the linked data representation of all the information involved in the process so as to make it searchable through semantic queries. Regarding the propaganda magazines examples, data was structured as follows:

- Dabiq and Rumiyah are magazines directed by the Islamic State.
- Each edition of both magazines are distributed as issues.
- The textual information is extracted using GSI Crawler. This is annotated using the PROV-O ontology.
- Each text is analyzed, and the places, people and organizations appearing on it are also described. This is annotated using NIF ontology, and the extracted entities are categorized according to Schema.org specifications.
- Finally, each concept is linked with DBPedia entities if possible.

Here is an example of the result, showing an article from the fifth issue of Rumiyah magazine. First, we define with PROV-O a scraping activity, which is the one used for extracting the information. Then, Rumiyah magazine is defined as a PROV-O agent and linked with DBPedia, and its fifth issue with the URI used for its publication, annotated as a Periodical schema instance being part of Rumiyah magazines.

Listing 3.1: Magazine and issue annotation

```
:scrapingActivity a prov:Activity .

<http://dbpedia.org/resource/Rumiyah_(magazine)> a prov:Agent .

<http://(...)/resources/Rumiyah5> a schema:Periodical ;
     schema:isPartOf <http://dbpedia.org/resource/Rumiyah_(magazine)> .
```

Now we show an example of an article annotation. It is defined by its URI, being an instance of type Article according to Schema ontology, with other Schema properties such as a headline, its article body (which has been shortened in the image) and its author. Additionally, its origin and how it was obtained are annotated using PROV-O once again.

Below the article, we can see an example of how additional information extracted from the article is annotated. In this case we can see an organization present in the article, which has been annotated using NIF, describing in which place of the text is located the referred entity, its type and its linkage with DBPedia.

Source	Issues	Articles	Organizations	Places	People	Triples
Dabiq Magazine	14	148	275	957	575	1955
Rumiyah Magazine	12	122	199	421	842	1596
Total	26	270	474	1378	1417	3551

Table 3.1: Magazine results

Listing 3.2: Magazine Article and entity annotation

```
<http://trivalent.cluster.gsi.dit.upm.es/resources/Rumiyah5-Collateral-
    Carnage> a schema:Article ;
     schema:articleBody """Allah revealed the Shariah to the Prophet,
         giving  people a complete way to live their lives. Unlike man-
         made systems, the law of the  Shariah is divine and flawless.
         There is no doubt in its authority and no  suspicion of its
         perfection. Allah said, We have not neglected anything in the
         Book (Al-Anam 38)(...)""" ;
     schema:author <http://dbpedia.org/resource/ISIL> ;
     schema:headline "Collateral Carnage" ;
     schema:pageStart "5" ;
     prov:wasDerivedFrom <http://trivalent.cluster.gsi.dit.upm.es/
         resources/Rumiyah5> ;
     prov:wasGeneratedBy :scrapingActivity ;


<trivalent.cluster.gsi.dit.upm.es/resources/Rumiyah5-Collateral-Carnage#
    offset_2451_2462_> nif:anchorOf "Abu Hanifah" ;
     nif:beginIndex 2451 ;
     nif:endIndex 2462 ;
     itsrdf:taClassRef schema:Organization ;
     itsrdf:taIdentRef <http://dbpedia.org/resource/Abu_Hanifa> .
```

Lastly, the Table 3.1 shows the linked data information obtained as a result of the scraping and analysis processes.

Regarding newspapers, each article was annotated the same way as articles extracted from propaganda magazines which has just been described. In this case, articles are not grouped into issues, and they are extracted in a dynamic basis, so the statistics are variable. Table 3.2 summarizes the statistics of scraped information:

Source	Articles	Triples
CNN, NYT, AJ	100/month	300/month

Table 3.2: Newspaper results

Listing 3.3: Newspaper Article and entity annotation

```
 <http://trivalent.cluster.gsi.dit.upm.es/resources/cnn1 a schema:Article>
      schema:ArticleBody "A decision by (...)";
      schema:author "<http://dbpedia.org/resource/cnn>";
      schema:headline "What's a stake for (...)" ;
      schema:thumbnailUrl "<thumbnailUrl.jpg>";
      schema:datePublished "2018-04-09";


<http://trivalent.cluster.gsi.dit.upm.es/resources/cnn1#offset_14_26 nif:
    anchorOf "Donald Trump">
      nif:beginIndex 14;
      nif:endIndex 26;
      Itsrdf:taClassRef schema:Person;
      Itsrdf:taIdentRef <thttp://dbpedia.org/resource/Donald_Trump;
      schema:thumbnailUrl "<thumbnail.jpg>";
```

3.2 Scraping

This section refers to the first process over the whole workflow, which is information gathering. As information is extracted from unstructured sources, this process is known as scraping. The said information sources were classified into three categories, but concerning their extraction method we consider two different categories: from web or PDF sources. Online newspapers are in the web sources categories, along with social networks, with the difference that social networks offer their own API to gather their data. On the other hand, propaganda magazines are found on PDF format and so the scraping techniques are different.

Thanks to GSI Crawler platform, it is possible to execute scraping tasks in a periodical form controlled by a cron tasks manager. This is useful for having a completely automated monitoring system which looks for new data every day or in a desired time window. For example, news could be gathered in a daily basis as well as tweets, or even following hourly

rates. On the other hand, propaganda magazines aren't updated in such regular way, and they should be first downloaded from their location and then ingested into the system.

3.2.1 Web Scraping

Web scraping is a technique to extract information from web sites which involves several levels of automation. This field has the following characteristics and methodologies: use of the HTTP protocol to access the desired server, use of regular expressions for parsing the HTML file, use of other languages such as XQuery also for parsing the HTML content, data mining algorithms in order to detect certain patterns or information recognition through semantic procedures extracted from annotated metadata.

In this section we use web scraping techniques for extracting information from international online newspapers: CNN News, The New York Times and Al-Jazeera. Our goal is to extract information related with jihadism, and so we have chosen newspapers with International relevance and one of them (Al-Jazeera) specialized in the Middle-East. A possible use case could be for example monitoring news about "ISIS".

All the developed web scrapers output their information in Linked Data Format, as stated in the previous section.

3.2.1.1 CNN

CNN News is a news channel from the US with international relevance in the scene. For extracting news items from it we can use their search functionality, which allows us to search and filter according to a certain topic, as shown in Figure 3.2. The news items appearing on the response of that query are the ones we are interested in, so we first need to obtain their links.

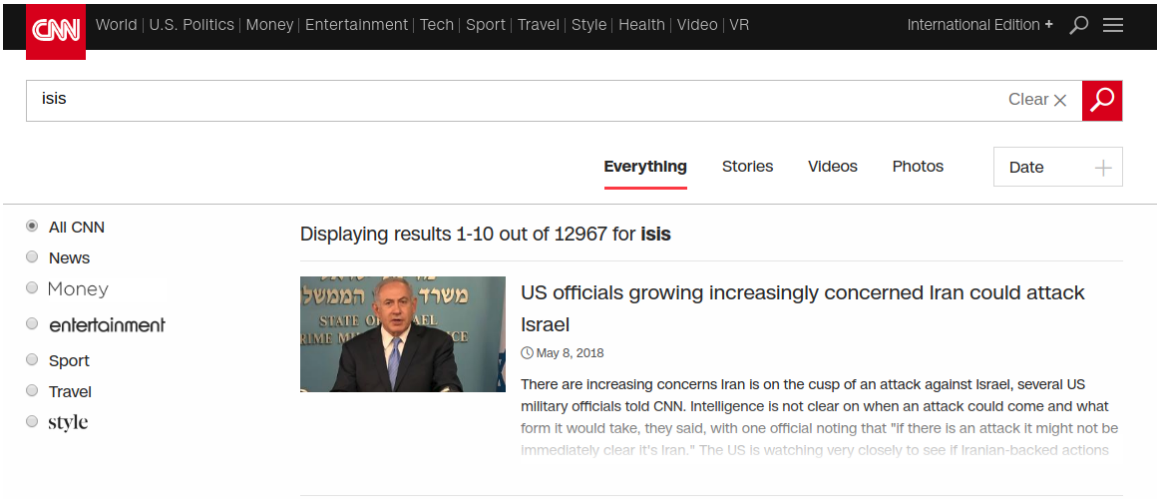


Figure 3.2: CNN list of filtered news items.

To extract news from its web, we first noticed that the search responses were not initially loaded in the HTML file but present in another response file returned from a news search API as shown in Figure 3.3.

Figure 3.3: CNN response.

From the obtained result field we can iterate through every news item, which contains all the information we need to fill our output.

3.2.1.2 The New York Times

The New York Times is also a news channel from the US with international coverage. Figure 3.4 shows its search page.

In this case, we have a similar situation than before, as we need to access an API which can also be found through the Developer Tools console. However, the body of each news item is not present in the response, and so we have to use the Newspaper Python library to extract it. This library takes the extracted URLs as inputs and returns the body of each news item, and so it is possible to get the same output form as in the CNN case.

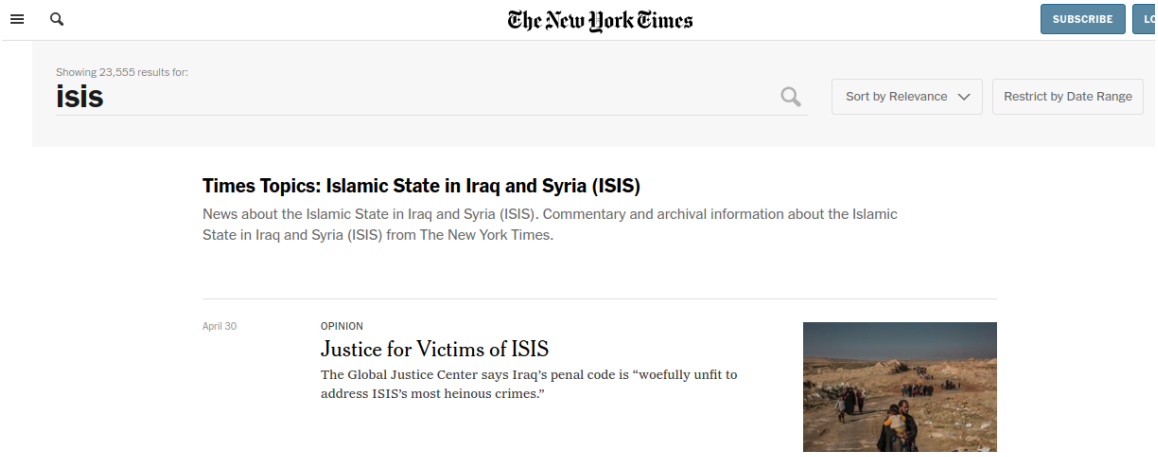


Figure 3.4: NY Times list of filtered news items.

3.2.1.3 AlJazeera

AlJazeera is the principal news channel of the arab world and one of the biggest ones worldwide with an audience of more than 270 million homes. Figure 3.5 shows its search page.

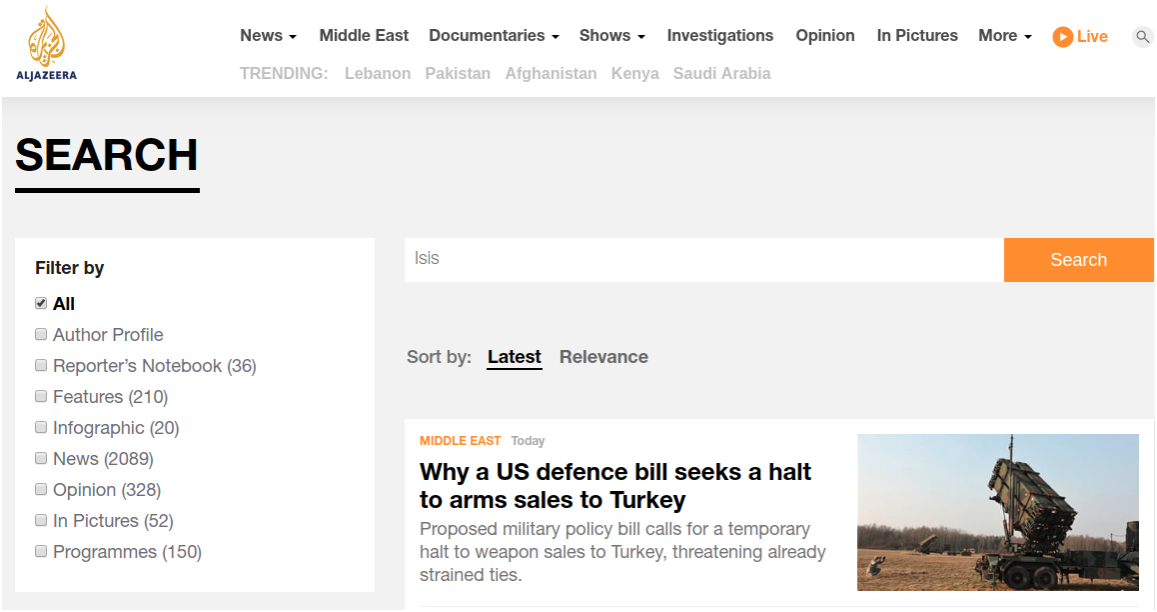


Figure 3.5: Al Jazeera list of filtered news items.

Once again, the goal was getting URLs from the news related to a given search term. However, this operation was harder than the ones before. The information we want to obtain is not sent in the HTML file from the beginning, but in this resource that we can find by inspecting the developer tools of our browser as shown in Figure 3.6..

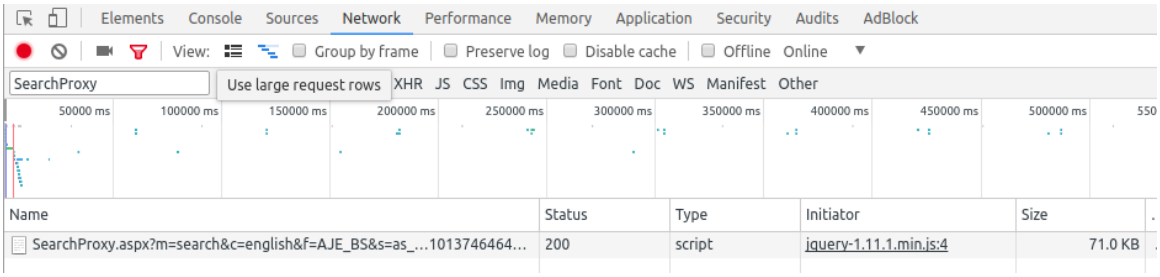


Figure 3.6: AlJazeera relevant resource.

Once the right endpoint was identified, we inspected its response content. However, it is not easy to extract information from it and so we have to make use of regular expressions. From this file we obtain both the article URLs and their headlines. Figure 3.7 shows the relevant information to be extracted.

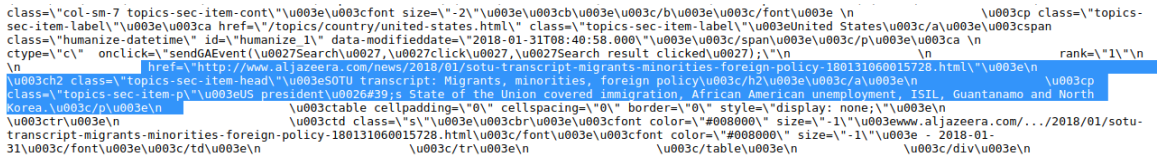


Figure 3.7: AlJazeera relevant resource content.

Using the URLs which were previously extracted, we can now obtain the desired information using the newspaper Python library. In this case, the headlines weren't extracted correctly using this library, and so we had to obtain them manually also with regular expressions.

3.2.2 PDF Scraping

PDF scraping consists on gathering information contained in PDF files. It is known as a hard task due to the lack of understandable tags which is present in other sources such as HTML pages. However, it is possible to use some libraries to execute queries over the content of PDF files.

For this project, there have been used two different Python libraries:

1. **Pdfquery** [8], which allows us to use JQuery [26] or XPath [24] syntax in order to gather information from PDF files. Additionally, it is possible to transform PDF files into a XML syntax in order to facilitate the visualization of the content structure for designing the right queries.

2. **Pdfminer** [7], which enables to extract the textual information of PDF files, even filtering by page number, and also enables to extract the PDF table of contents when

possible (some PDF files don't contain this information).

All the developed PDF scrapers output their information in Linked Data Format, as stated in the Data Modelling Section.

3.2.2.1 Dabiq

Dabiq was an online magazine used by the Islamic State of Iraq and the Levant (ISIL) for Islam and recruitment. It was published by ISIL via the Deep Web, but could also be found through ordinary web sources. Its issues were published between July 2014 and July 2016, existing a total of 15 editions.

For gathering Dabiq content information we have first used pdfquery in order to extract the XML translation of each issue in PDF format. Figure 3.8 shows the XML resultant file:

<pdfxml CreationDate="D:20150713205757+03'00'" Creator="Adobe InDesign CC (Macintosh)" ModDate="D:20150713205906+03'00'" Producer="Adobe PDF Library 10.0.1" Trapped="/False">
  <LTPage bbox="[0, 0, 595.276, 841.89]" height="841.89" pageid="1" rotate="0" width="595.276" x0="0" x1="595.276" y0="0" y1="841.89" page_index="0" page_label="">
    <LTImage bbox="[0.076, -0.03, 595.276, 841.89]" bits="8" colorspace="["/DeviceCMYK"]" height="841.92" imagemask="null" name="Im0" srcsize="[2480, 3508]" stream="<PDFStream(1410): raw=5390345, {'BitsPerComponent': 8, 'ColorSpace': /'DeviceCMYK', 'DecodeParms': {'BitsPerComponent': 8, 'Colors': 4, 'Columns': 2480}, 'Filter': /'FlateDecode', 'Height': 3508, 'Intent': /'RelativeColorimetric', 'Length': 5390344, 'Metadata': <PDFObjRef:1409>, 'Name': /'X', 'Subtype': /'Image', 'Type': /'XObject', 'Width': 2480}>" width="595.2" x0="0.076" x1="595.276" y0="-0.03" y1="841.89">
      <LTFigure bbox="[0.076, -0.03, 595.276, 841.89]" height="841.92" matrix="[595.2, 0.0, 0.0, 841.92, 0.076, -0.03]" name="Im0" width="595.2" x0="0.076" x1="595.276" y0="-0.03" y1="841.89"/>
    </LTImage>
  </LTPage>
  <LTPage bbox="[0, 0, 595.276, 841.89]" height="841.89" pageid="2" rotate="0" width="595.276" x0="0" x1="595.276" y0="0" y1="841.89" page_index="1" page_label="">
    <LTImage bbox="[0.105, -0.015, 595.305, 841.905]" bits="8" colorspace="["/DeviceCMYK"]" height="841.92" imagemask="null" name="Im0" srcsize="[2480, 3508]" stream="<PDFStream(22): raw=403064, {'BitsPerComponent': 8, 'ColorSpace': /'DeviceCMYK', 'Filter': /'DCTDecode', 'Height': 3508, 'Intent': /'RelativeColorimetric', 'Length': 403063, 'Metadata': <PDFObjRef:21>, 'Name': /'X', 'Subtype': /'Image', 'Type': /'XObject', 'Width': 2480}>" width="595.2" x0="0.105" x1="595.305" y0="-0.015" y1="841.905">

Figure 3.8: Dabiq PDF as XML.

In this case, the approach followed to extract the table of contents was directly reading the contents from the generated XML file. An interesting finding was that the textual information was the only content between tag endings and tag beginnings. Thanks to this, the processing was done as follows. All the XML file has a tag structure in which each tag contains its own parameters. However, the relevant finding was that written text on the PDF files was represented as raw text between tag endings and beginnings. Figure 3.9 shows this finding.

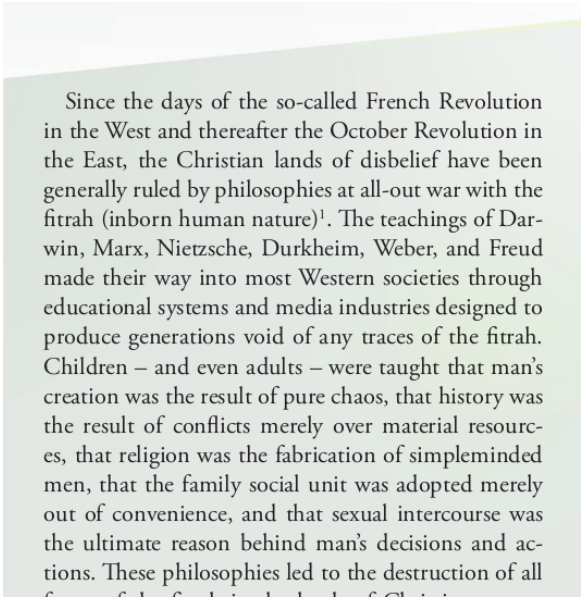
      <LTTextLineHorizontal bbox="[121.633, 623.318, 204.283, 643.598]" height="20.28" width="82.65" word_margin="0.1" x0="121.633" x1="204.283" y0="623.318" y1="643.598"><LTTextBoxHorizontal bbox="[121.633, 623.318, 204.283, 643.598]" height="20.28" index="1" width="82.65" x0="121.633" x1="204.283" y0="623.318" y1="643.598">F O R E W O R D <Annot A="<PDFObjRef:23>" BS="{"S": "/S", "Type": "/Border", "W": 0}" Border="[0, 0, 0]" H="/N" Subtype="/Link" Type="/Annot" bbox="[121.638, 643.593, 204.278, 623.32]" height="-20.27299999999991" width="82.63999999999999" x0="121.638" x1="204.278" y0="643.593" y1="623.32"/></LTTextBoxHorizontal></LTTextLineHorizontal>

Figure 3.9: Text information in Dabiq as XML .

Finally, text extraction was made through regular expressions to extract the text between said tags. The aim of the first step was extracting the articles present on each magazine and their starting page.

Once the table of contents was extracted, each article was extracted with the Pdfminer library, which enables to extract the content of each page, being possible to select the range of pages to read. As a side note, it was not possible to extract the table of contents this way as the resultant information wasn't returned in a organised way.

Figure 3.10 shows an example of an extract from a Dabiq magazine, and Figure 3.11 the output of the scraper.



Since the days of the so-called French Revolution in the West and thereafter the October Revolution in the East, the Christian lands of disbelief have been generally ruled by philosophies at all-out war with the fitrah (inborn human nature)[1]. The teachings of Darwin, Marx, Nietzsche, Durkheim, Weber, and Freud made their way into most Western societies through educational systems and media industries designed to produce generations void of any traces of the fitrah. Children – and even adults – were taught that man's creation was the result of pure chaos, that history was the result of conflicts merely over material resources, that religion was the fabrication of simpleminded men, that the family social unit was adopted merely out of convenience, and that sexual intercourse was the ultimate reason behind man's decisions and actions. These philosophies led to the destruction of all

Figure 3.10: Dabiq example extract.

```
Since the days of the so-called French Revolution in the West and thereafter the October Revolution in the East,
the Christian lands of disbelief have been generally ruled by philosophies at all-out war with the fitrah (inborn
human nature)1. The teachings of Dar- win, Marx, Nietzsche, Durkheim, Weber, and Freud made their way into most W
estern societies through educational systems and media industries designed to produce generations void of any tra
ces of the fitrah. Children – and even adults – were taught that man's creation was the result of pure chaos, tha
t history was the result of conflicts merely over material resourc- es, that religion was the fabrication of simp
leminded men, that the family social unit was adopted merely out of convenience, and that sexual intercourse was
the ultimate reason behind man's decisions and ac- tions. These philosophies led to the destruction of all facets
of the fitrah in the lands of Christian pagan- ism. They destroyed the basis of religiosity – albeit a corrupt on
e fashioned from paganism and tarnished scripture – and what it entailed of morality and soci- ety.
```

Figure 3.11: Dabiq scraped information.

3.2.2.2 Rumiyah

Rumiyah is also an online magazine used by the Islamic State of Iraq and the Levant for propaganda and recruitment which was first published in September 2016. It is released in several languages including English, French, German, Russian or Indonesian. It replaces Dabiq magazine, and this change of name could be due to the loss of the town of Dabiq by ISIL according to some analysts. Its last issue was released on September 2017, making a total of thirteen which were released at an approximately monthly rate.

In this case it is possible to extract the table of contents from the Pdfminer library, as Rumiyah PDFs contain this information in each issue.

Once the table of contents is extracted that way, the text extraction of each article is made the same way as Dabiq's. Figure 3.12 shows a table of contents example, and Figure 3.13 its conversion into XML file, being noticeable that page numbers are one value lower than the original ones.

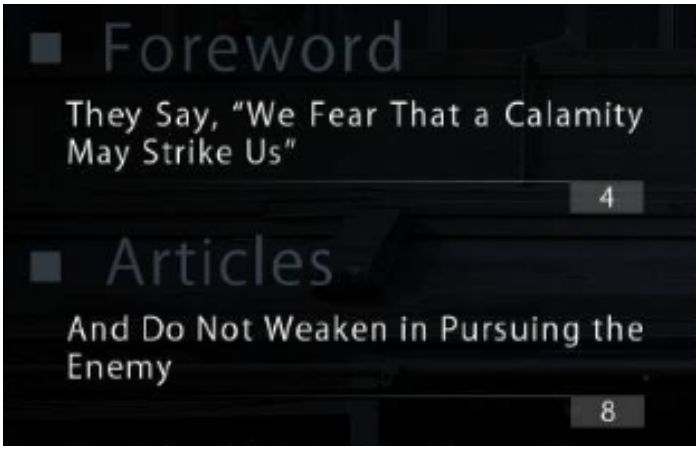


Figure 3.12: Rumiyah table of contents example.

```
<outlines>
  <outline level="1" title="Cover">
    <dest>
      <list size="2">
        <ref id="229"/>
        <literal>Fit</literal>
      </list>
    </dest>
    <pageno>0</pageno>
  </outline>
  <outline level="1" title="They Say, "We Fear That a Calamity May Strike Us"">
    <dest>
      <list size="2">
        <ref id="11"/>
        <literal>Fit</literal>
      </list>
    </dest>
    <pageno>3</pageno>
  </outline>
  <outline level="1" title="And Do Not Weaken in Pursuing the Enemy">
    <dest>
      <list size="2">
        <ref id="28"/>
        <literal>Fit</literal>
      </list>
    </dest>
    <pageno>7</pageno>
  </outline>
```


Figure 3.13: Rumiyah extracted table of contents example.

3.3 Data Analysis and Enrichment

This section shows how the previously gathered data obtained as the output of the scraping task can be enriched through diverse analysis modules which are deployed as Senpy plugins. Additionally, GSI Crawler enables the concatenation of those processes through the so called Pipelines, which allow the flow of information since its collection augmenting it with Senpy analysis. Senpy provides a modular solution for developing and deploying pieces of software with the aim of performing analysis tasks outputting responses in linked data format.

In order to accomplish the Trivalent project tasks, two analysis tasks were required to be used by the GSI Crawler solution tool, which were developed as Senpy plugins. Senpy plugins are the modular form of software services, which provide added value capabilities for data analysis tasks, easing their implementation and deployment thanks to Senpy architecture.

In addition to the development of those plugins, a Senpy Playground was deployed in order to make them publicly available. Senpy Playground is a site where Senpy Plugins are deployed and so can be accessed using a web browser. It provides a nice graphical user interface for testing plugins and also the endpoint of each plugin so as to use it in a programmatical way.

This section first defines the new plugins developed in this thesis then some examples of pipelines which were used.

3.3.1 BING translator

This plugin translates any text using the Microsoft Translator API. The idea of creating this plugin is the amount of useful information of this project which is written in Arabic language.

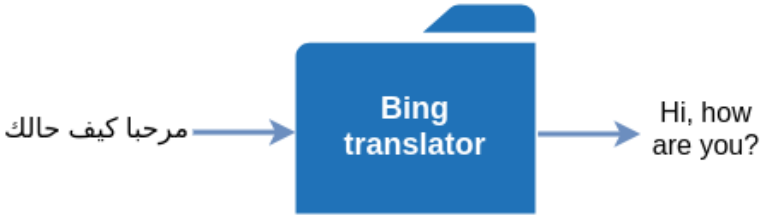


Figure 3.14: Translator plugin.

The following images illustrate the behaviour of the plugin accessed using Senpy Playground.

- We want to analyze the following text with the Bing plugin. We paste it into the text box appearing on it, select the plugin we want to use and we introduce our API key.

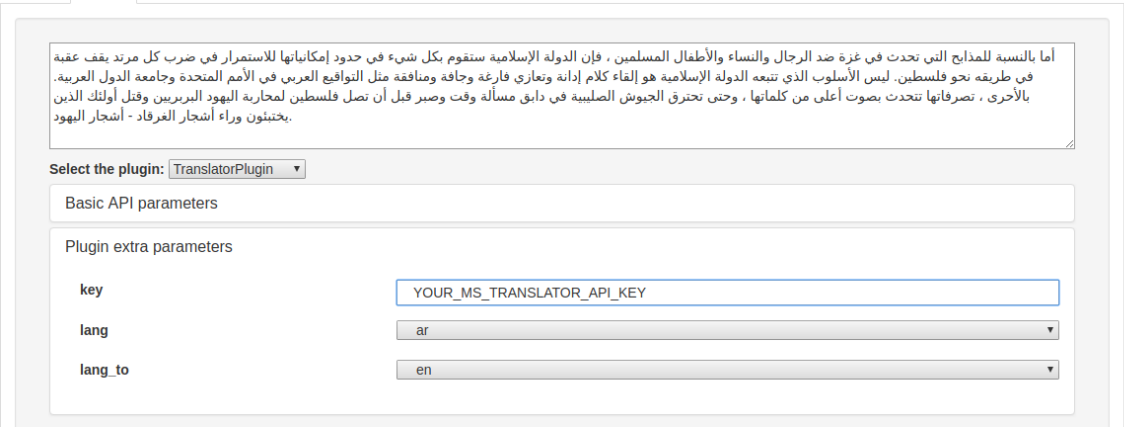


Figure 3.15: Introducing information.

- As a result, we obtain the following output.

```
nif:isString "As for the massacres taking place in Gaza agains the muslim men, women and children
```

Figure 3.16: Translator output.

3.3.2 COGITO Plugin

COGITO [1] is a software designed by Expert System oriented to taking advantage of human intelligence in cognitive computing. This proprietary software is able to read, understand and answer the necessities of the users in order to improve and enrich decision making at high speed in a multilingual way.

COGITO bases its cognitive computing system in machine learning algorithms with the following features:

- It contains default knowledge, being based in a wide and deep knowledge representation ready to use thanks to their language mapping called *Sensigrafo* (an ontology containing millions of word definitions, related concepts and so more).
- It can read and understand in the way a human does, breaking the barriers of ambiguity and identifying the right meaning of words and expressions according to their context.
- It transforms the obtained knowledge in actionable text, being able to emulate some human comprehension processes. This fact enables the detection of patterns, signals and existing data connections.
- It learns from human experience, augmenting its intelligence from human experts and acquiring new knowledge from written communications.

This plugin uses the COGITO Expert System API for extracting entities from texts. The extracted entities can be organizations, places and people. Additionally, it also returns if a text belongs to certain classes of a terrorism and crime taxonomy.

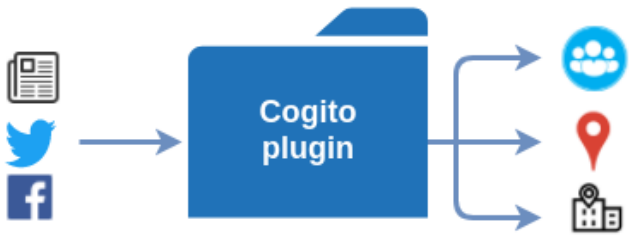


Figure 3.17: COGITO plugin.

The following images show an entity recognition example, using the output of the previous example. This pipeline structure can be easily achieved using GSI Crawler to concatenate Senpy plugins.

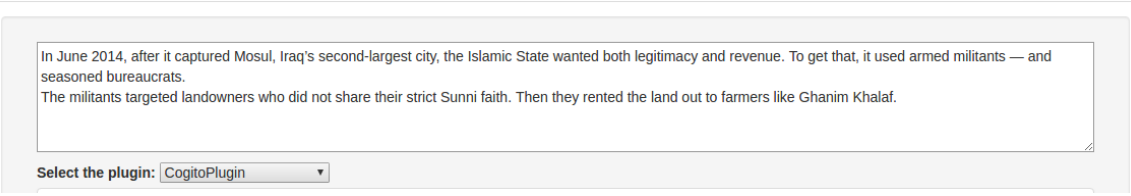


Figure 3.18: Introducing information.

- First, we introduce the text we want to analyze.
- The response includes information about the entities which were found on it.

```
:organizations [ a ns1:Organization ;
        ns1:name "ISIL" ] ;
:people [ a ns1:Person ;
        ns1:name "Ghanim Khalaf" ] ;
:places [ a ns1:Place ;
        ns1:name "Mosul" ],
    [ a ns1:Place ;
        ns1:name "Iraq" ] ;
```

Figure 3.19: COGITO output.

3.3.3 Custom pipelines

As it has been stated before, GSI Crawler allows us to create custom pipelines in order to carry out certain tasks. In this section we show some of the offered solutions which have been deployed for Trivalent project.

Our first example shows a pipeline combining the previously explained plugins, which were developed for Trivalent project. In this case, the input text is written in Arab language and the result is its translation enriched with the extracted organizations, places and people.

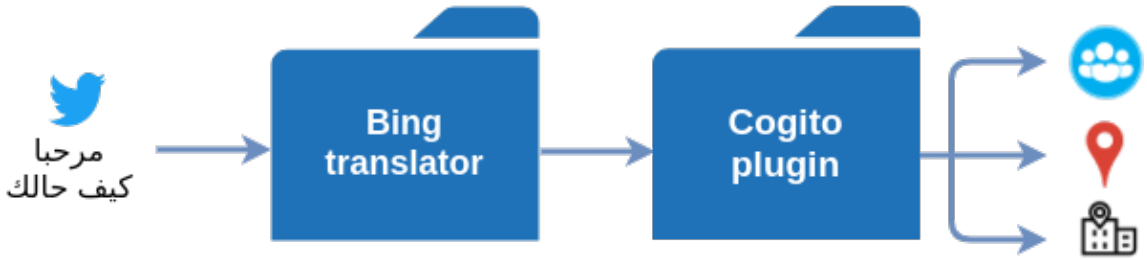


Figure 3.20: Pipeline Bing + COGITO.

The pipeline can also be augmented by a storage process. GSI Crawler offers the possibility of easily storing data on ElasticSearch so as to persist analysis in order to avoid redundancy of processes. For example, if a tweet has been previously crawled, it is not necessary to repeat the process in the future but the result is kept.

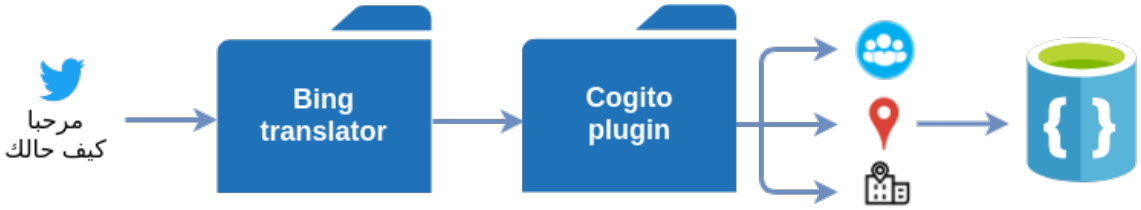


Figure 3.21: Pipeline Bing + COGITO + ElasticSearch.

We can also add elements at the beginning of the pipeline. In the previous example we directly obtained the source information from Twitter API, but in this case we show the possibility of integrating a PDF scraper to all the pipeline (PDF scraping is described in the following chapter). The goal of this pipeline is being able to extract information from PDF documents, transform them into a textual version, and performing the desired operations. In this case, entity extraction is performed once again and the result is also stored in ElasticSearch.

Figure 3.22: Pipeline PDF Scraper + COGITO + ElasticSearch.

3.3.4 Use case: Additional information extracted from Dabiq and Rumiyah

Thanks to the use of the COGITO plugin, it was possible to extract entities appearing on both propaganda magazines including organizations, people and places. This subsection shows an analysis of the extracted entities.

Table 3.3 shows the most mentioned organizations in both Dabiq and Rumiyah magazines. As it can be seen, the majority of those most mentioned organizations are shared between both magazines as Rumiyah is a second version of Dabiq. The most frequent organization is Islamic State of Iraq and the Levant (ISIL) followed by Party of Allah in both cases. However, Al-Qaida presence is highly reduced in Rumiyah. Khilafah is a top frequent word in both as great part of their narratives are centered around it.

After seeing the comparison between the most frequent people entities, we do the same comparison for people. As before, some of them are present in both magazines as the most frequent ones and Al-Bukhari is the most frequent person, a Persian Islamic scholar who was born in Bukhara. Table 3.4 shows the obtained results.

Lastly, we show the extracted places from both magazines. Those places were once again obtained with the COGITO plugin and enriched using the Google Maps Geocoding API [2]. This API gives valuable information such as address components and geolocation. For this analysis, it was valuable to obtain the coordinates of each location extracted so as to plot them in a map.

For plotting the information on a map, it was necessary to incorporate the Basemap toolkit to the matplotlib Python library.

Figure 3.23 shows the global locations appearing on Dabiq (red) and Rumiyah (magazines).

There are a larger number of place references in Rumiyah, but the main difference are the references to Indonesia which are significantly higher in Dabiq Magazines. When zooming the most mentioned area in the Middle East, the result shown in Figure 3.24 is obtained.

Magazine	Organization	Ocurrences
Dabiq	ISIL	348
	Party of Allah (Hezbollah)	59
	Al-Qaida	48
	Khilafah	33
	Free Syrian Army (FSA)	28
	Kurdish Workers Party	22
	Islamic Emirate of Afghanistan	18
	Rafidah	18
	Nusayri	14
	NASA	14
Rumiyah	ISIL	137
	Party of Allah (Hezbollah)	70
	Kurdish Workers Party	25
	Khilafah	22
	Rafidi	19
	Turkish Land Forces	16
	Islamic Emirate of Afghanistan	14
	Afghan National Army	11
	Nusayri	9
	Rafidah	6

Table 3.3: Most Frequent Organizations

Magazine	Person	Ocurrences
Dabiq	Al-Bukhari	32
	Barack Obama	25
	Ali Ibn Abi Talib	18
	Abu Dawud	20
	Abu Bakr al-Baghdadi	12
	Jawlani	12
	Mulla Umar	12
	Abdullah Ibn Al-Imam Ahmad	11
	Ibn Majah	10
	Abu Ibrahim	10
Rumiyah	Al-Bukhari	71
	Siyyid ‘Alí Muhammad	19
	Abu Hurayrah	18
	Ibn ’Abbas	17
	Abu Dawud	16
	Ibnul-Athir	11
	Anas Ibn Malik	10
	Abu Hamzah	9
	Ibn Kathir	9
	Abdullah Ibn Yasin	9

Table 3.4: Most Frequent People

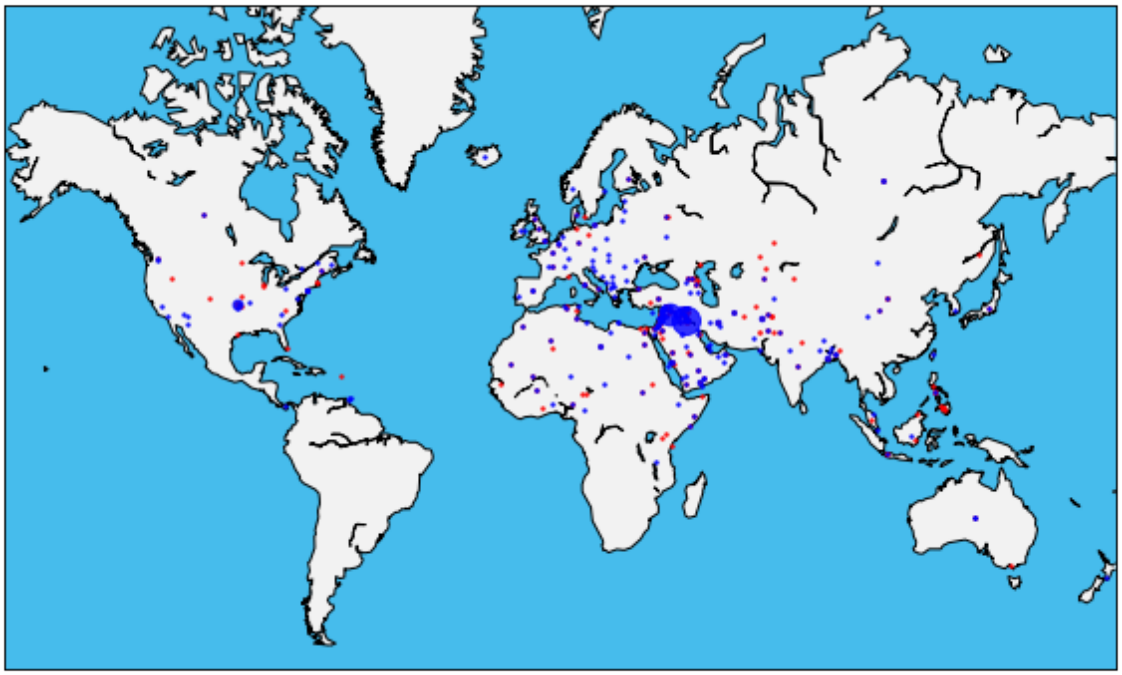


Figure 3.23: World locations mentioned in Dabiq (red) and Rumiyah (blue).

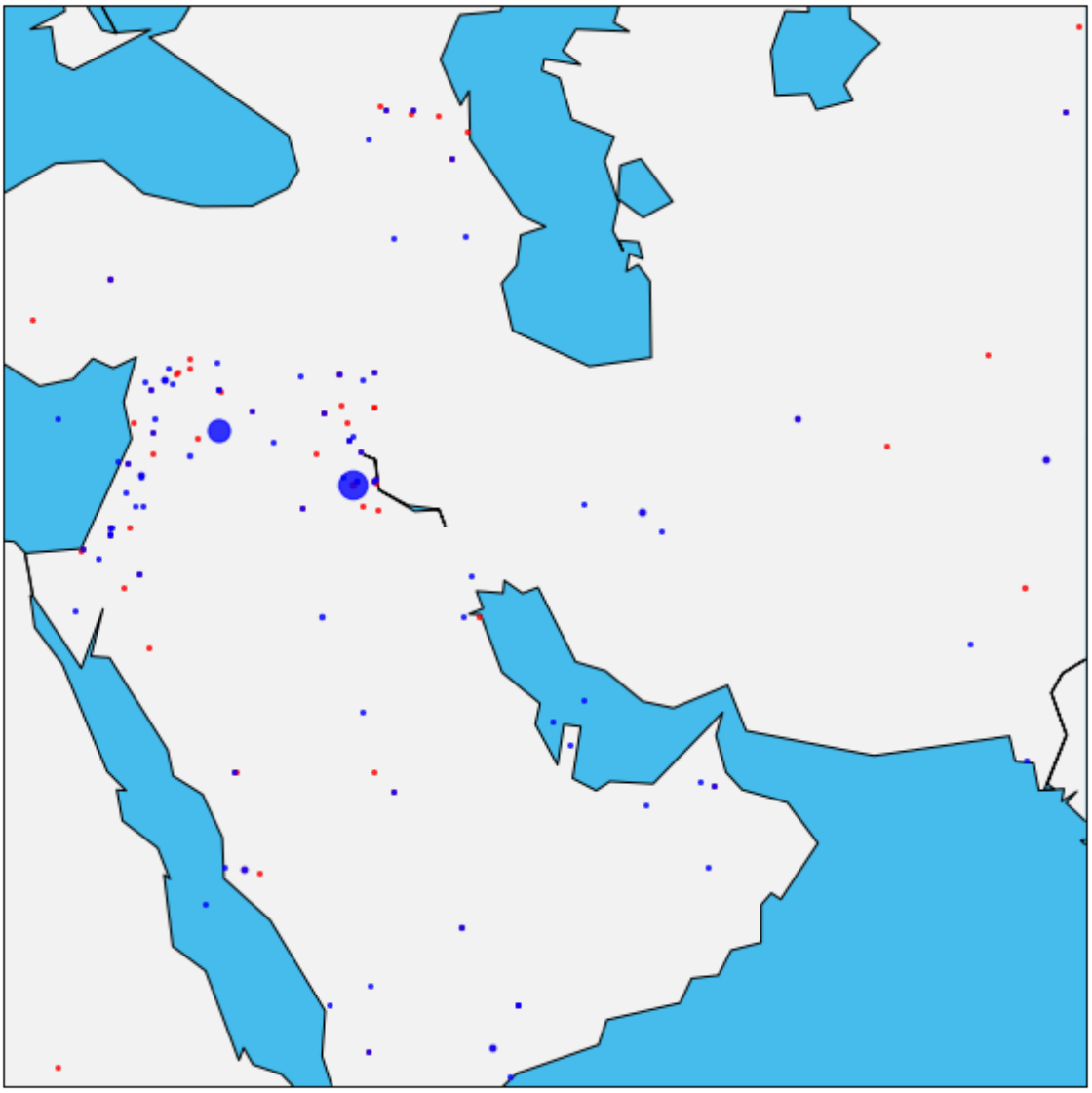


Figure 3.24: Middle East locations mentioned in Dabiq (red) and Rumiyah (blue).

3.4 Visualization

Up to here we have seen all the capabilities of the system, but it is also needed to visualize the obtained information. With that goal in mind, it has been developed a visualization layer in form of a dashboard. This dashboard consists of a web application based on the usage of Polymer Web Components, and obtains the data by connecting to the ElasticSearch instance in which the data has been stored by the defined pipelines. This section describes all the modules that can be found on it.

First, we can see a header containing all the instances of data extracted from each of the scraped sources: CNN News, The New York Times, Al Jazeera, Twitter and propaganda magazines directed by ISIS (Dabiq and Rumiyah). It is possible to filter results depending on their source. Figure 3.25 shows the described features.

Then, making use of the Senpy COGITO plugin for entity extraction, it is possible to display the extracted organizations, people and places. In the case of organizations, the most frequent ones are displayed along with their count as shown in Figure 3.25. The most mentioned people entities are shown in Figure 3.26, appearing also their associated photo extracted from DBpedia. Lastly, places appearing on each data source have been plotted over an interactive heat map as shown in Figure 3.27. Additionally, we can see the sentiment distribution along all sources, stating whether they have a positive, neutral or negative sentiment. It is also possible to see the temporal distribution of the collected sources, which can indicate that some relevant event has happened. Those two last features are also shown on Figure 3.26.

Finally, each gathered document is presented over a background representing its sentiment polarity (red for negative and green for positive). Additionally, each sentence has also been highlighted with its sentiment polarity colour and at the end of each text there is a display showing the extracted entities from it. Figure 3.27 shows this feature.

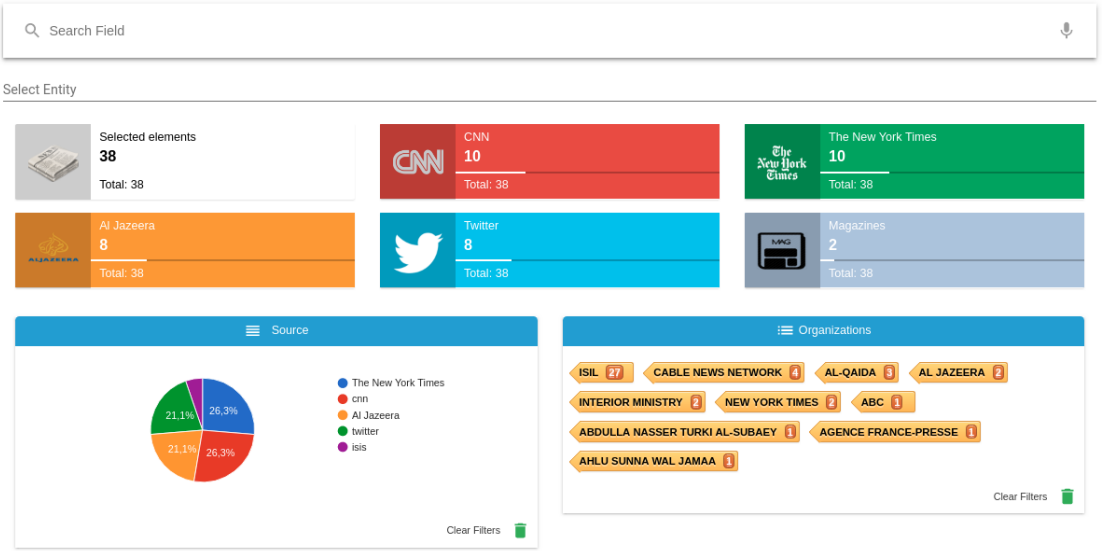


Figure 3.25: Available sources at the dashboard.

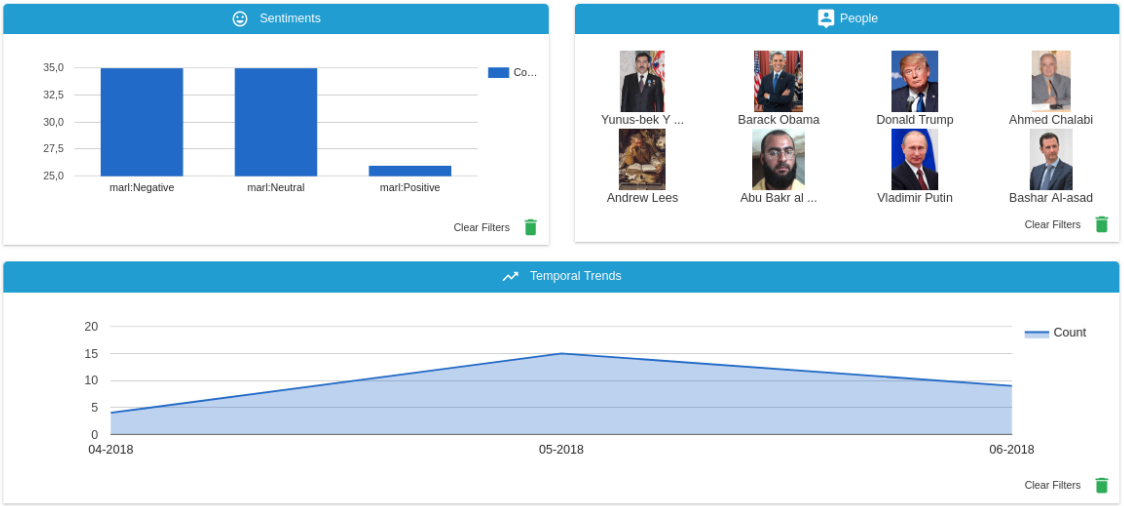


Figure 3.26: Most mentioned organizations and people.

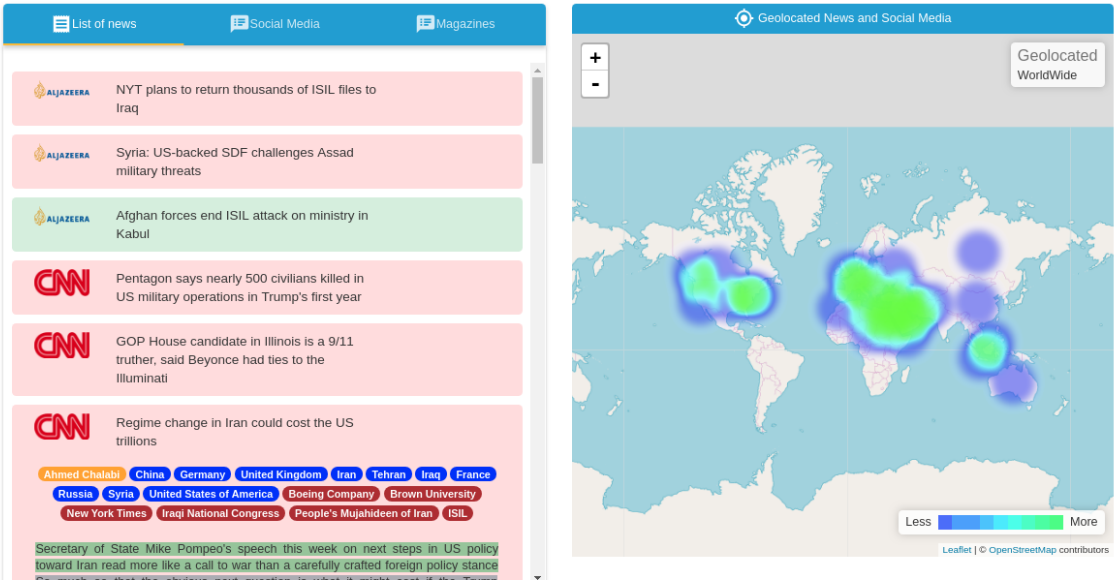


Figure 3.27: Geolocation of data sources.

3.5 Annotation

In the previous sections the data monitoring platform was described, stating its data ingestion capabilities along with the possibility of persisting it. Thanks to the definition of pipelines, it is possible to provide additional analysis to the obtained texts such as entity extraction. However, in this section we are describing another use case of the developed platform, which fulfills another requirement of H2020 project Trivalent goals.

The goal is creating an annotation platform for jihadist radical texts, with the aim of manually stating whether a text contains certain known jihadist narratives or not. Those narratives are defined by groups of experts who have studied more radical texts, and have been classified into a taxonomy. This manual annotation task is done with the purpose of creating an automatic system to detect those narratives in future texts, for example by machine learning means. For example, the following chapters show how to build a jihadist tweet classifier to detect radicalism using pre-annotated tweets for its development. The same process could be done for automatically detecting specific narratives appearing over any text.

The texts to be annotated are the articles extracted from Dabiq and Rumiyah magazines, as they are produced by the Islamic State with the aim of joining people to their doctrine. As they were gathered with GSI Crawler and stored in ElasticSearch, it is possible to show them. For the annotation task, it was needed to provide the possibility of selecting a text from a web page and labelling it according to certain parameters. For this task another tool was required, called Hypothes.is [4]. This tool allows us to select any text on a web page, add any annotation to it, and storing it associating the annotation to the web page's URI. So, our solution was showing the article on the URI where we decided to publish it so as to make the annotation referring to its correspondent article.

The integration of Hypothes.is[4] with any webpage can be done inserting javascript code on it. The results are shown in the following image, in which an article extracted with our data monitoring system is displayed. The annotation mode can be achieved by selecting the desired text and clicking the annotate button. Any partner of the project could use this tool for manually annotating narratives present at the gathered documents, and the results could be used for later analysis and counter-narrative design.

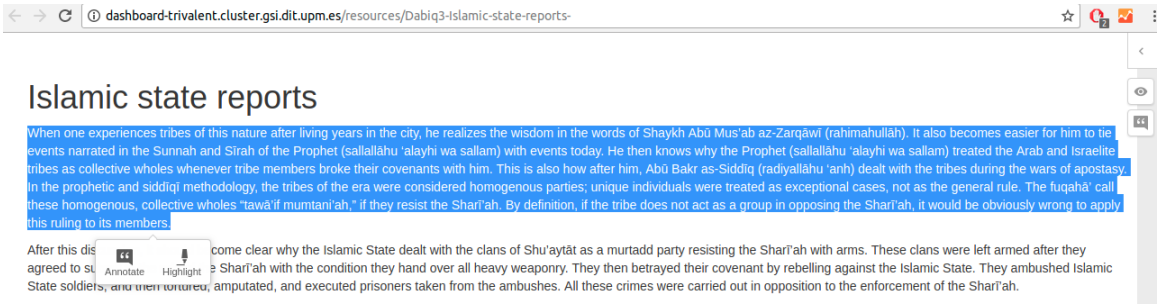


Figure 3.28: Narrative annotation with Hypothes.is.

3.6 Conclusions

This chapter has shown the development for this thesis of a Radicalist Monitoring System following Linked Data principles. First of all, the Linked Data modelling of the obtained information has been described. This modelling allows to perform semantic queries over the resulting dataset and enriching it thanks to the possibilities offered by Linked Data, as resources are identified by their URI.

For extracting the information, several steps have been made: first, a data ingestion module able to extract information through web and PDF scrapers has been developed. Then, analysis processes have been made over the obtained information, enriching the gathered information following once again Linked Data principles. Finally, the resultant dataset has been persisted, allowing its usability for other actions and its visualization. For visualizing the information, a dashboard based on Polymer Web Components has been developed aiming to show the different analysis performed over the gathered data. Additionally, an annotation capability has been granted for the platform in order to perform further experiments with the obtained dataset.

CHAPTER 4

Development of a Radicalisation Classifier

This chapter shows the experiments carried out to create a tweet radicalisation classifier based on Deep Learning techniques. The data monitoring platform developed for this project could serve as a tweet collector for the experiments and additional information extracted from its analysis can also be incorporated to the models in order to make them more accurate.

4.1 Dataset

The dataset used for the development of the classification model contains the Twitter timelines of 1132 European Twitter users. Those user timelines were labeled taking into account if they shared incitement material from known pro-ISIS accounts and also if they used extremist language. The result is a balanced dataset according to the user count of each class, with 566 pro-ISIS and anti-ISIS users, but unbalanced considering the number of tweets belonging to each class (the average number of tweets per anti-ISIS user is higher than in the pro-ISIS case).

As the number of users is too small for developing the desired experiments, the initial approach is done at tweet level, balancing the available tweets. As said before, the anti-ISIS case contains a lower number of tweets and so an additional operation was needed for balancing the number of instances of both classes. This dataset finally contained 1205020 instances, 602510 belonging to each class.

4.2 Methodology

For carrying out the development of a radicalisation classifier, the following steps have been followed. The goal is developing the model which obtains best F-Score over the test set without overfitting the training set. Several models have been explored, following in all of them a similar methodology than the one explained here.

First of all, it was necessary to split the dataset in a logical way. For avoiding the unbalanced number of instances for each class, the decision was selecting the same name of radical and not radical instances. That way, it is easier to avoid overfitting over the training dataset which could lead to a bad generalization of the model.

Additionally, when training neural networks with Keras, it is possible to split the available dataset into two subsets: one for training and the other one for validation selecting their desired proportion. The metrics offered by Keras show both the F-Score over the training and validation datasets, and so it is possible to check if overfitting occurs (F-Score over training dataset would be clearly superior over validation's one).

Once the dataset is split as desired, it is time to apply transformations in order to make textual information understandable by computer algorithms. The goal of this step is transforming words into vectors which can be fed into neural networks, and these vectors are known as Word Embeddings. More advanced details about how Word Embeddings have been used will be shown in the next section, but we will first describe the preprocessing needed to map words to those vectors starting from having raw tweets.

The process of extracting words from texts is known as tokenization. The goal for

using Word Embeddings is associating each token with its correspondent vector. However, words appearing on texts are not always in a canonical form as written language contains variations such as contractions or upper case letters at different positions. For solving the capital letters issue, each word is converted into its lower case form, and for contractions, these are some examples of applied transformations.

Pattern	Transformation
what's	what is
've	have
can't	cannot
i'm	i am
're	are
'd	would
'll	will

Applying those transformations, it is easier to extract the meaning of the text. Other preprocessing tasks are for example identifying URLs or Hashtags, which are really common in Twitter. Once we obtain all the different tokens present at the dataset, each of them will have a numerical identifier. This identifier will map each token to its corresponding vector afterwards. The values of those vectors are obtained by transfer learning [43], which means storing knowledge gained while solving one problem and applying it to a different but related problem. It will also be explained on next section.

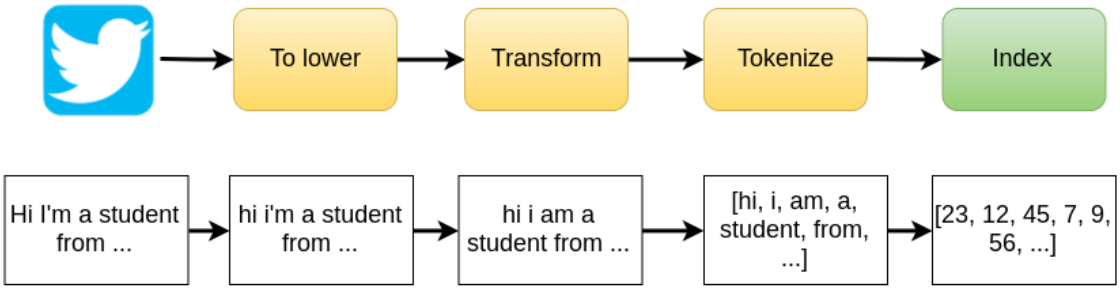


Figure 4.1: Tweet preprocessing.

Once we have the existing texts transformed into numerical vectors, it is possible to design different models and tune their hyperparameters in seek of the best F-Score, a metric which is explained as follows:

- True Positives (**TP**) are documents correctly classified as positive and True Negatives (**TN**) are documents correctly classified as negative.
- False Positives (**FP**) are documents wrongly classified as positive and False Negatives (**FN**) are documents wrongly classified as negative.
- **Precision** is the number of true positives divided by the sum of true positives and false positives, which is the number of positive results.

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

- **Recall** is the number of true positives divided by true positives and false negatives, that is to say, the number of positives which should have been predicted.

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

- **F1-Score** is the weighted average of precision and recall, ranging from 0 to 1.

$$F1 - score = \frac{2}{\frac{1}{P} + \frac{1}{R}} = 2\frac{P * R}{P + R} \tag{4.3}$$

This is a complex problem as there are too many variables to optimize, but by analyzing the results of different experiments it is possible to reach better solutions. Apart from the neural network architectures described on the previous chapter, more models have been explored. Additionally, several modifications have also been made due to some discoveries made during the first experimental attempts.

4.3 Word Embeddings

For developing these experiments, it was first necessary to obtain the word embeddings which were going to feed the entry of the neural networks. For this goal, there are two different options: generating our own word embeddings using the provided dataset or using pre-trained word embeddings. Both options have been explored in this project.

4.3.1 Pre-trained word embeddings

There are several pre-trained word-embeddings modules which have been opensourced, and one of the most famous ones is GloVe [45]. GloVe stands for Global Vectors for Word Representations, and they were obtained through an unsupervised learning algorithm performed on aggregated global word-word co-ocurrence statistics from a corpus. There are available various GloVe pre-trained word vectors which vary on their source (Wikipedia, Common Crawl...) and their embedding size.

4.3.2 Generating word embeddings

Having such a big corpus is suitable for generating our word embeddings. This can be done by using the Gensim library, which was previously explained on the Enabling technologies chapter. This Python library contains an implementation of the Word2Vec algorithm in which it is possible to choose the embedding dimension, a window size representing the context parameter of the algorithm and a minimum occurrence count for each word. For this project, 100-sized embedding vectors have been generated with a context size of five words and a minimum occurrence of four times for each word. Now, some interesting finding discovered while exploring the generated embeddings will be shown in order to prove their effectiveness.

4.3.2.1 Interesting findings with generated word embeddings

As the available dataset is classified into pro-isis and anti-isis accounts, it is possible to see some interesting differences between both groups. Despite words have their own meaning, they can be used in different ways due to additional connotations such as political factors. So, apart from creating the word embeddings using the whole dataset, additional experiments involved creating two different word embeddings, one per class, in order to explore their differences. Those embeddings are only going to be used for extracting the following information but not for doing the classification experiments lately (for this task, the one obtained with the whole dataset has been used).

Gensim library contains a suitable function for carrying out the following experiments which is finding the most similar words with respect to a given word according to their cosine similarity in the vector space. That way, it returns the words whose vectors have the highest cosine similarity with respect to the target.

The first experiments for showing those language relationships consist on displaying the nearest words to certain ones on a 2D plot. The initial vector shape was 100 components per vector, so visualization cannot be done. For plotting the word vectors in a suitable way, it was needed to apply a technique called t-SNE [38], which reduces the shape of word vectors to two.

Having a look at the results, there have been some interesting findings:

- The word *ISIS* in the anti-ISIS dataset has *terrorists* as one of the most similar words, but the same situation does not happen in the pro-ISIS case. This fact is shown in Figure 4.2.

- Other interesting finding similar to the other one happens when searching similar words to *USA*. In this case, in the pro-ISIS embeddings, the word invading appears

but not on the anti-ISIS case as shown in Figure 4.3.

- Figure 4.4 shows another surprising example is searching the word *oil*, which is closely related to *arms* also in the pro-ISIS tweets but not in the other case.
- Finally, when searching the term *western*, several negative words appear in the pro-ISIS embeddings such as *hostile*, *colonialism* and *corrupt*, as shown in Figure 4.5.

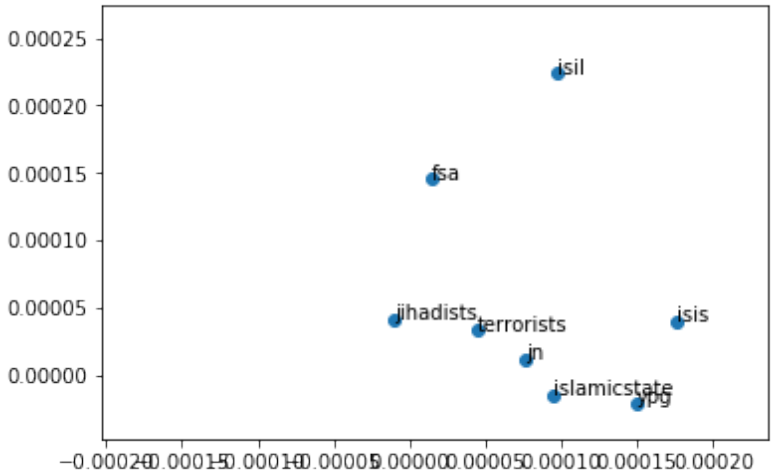


Figure 4.2: ISIS similar words in anti-ISIS tweets.

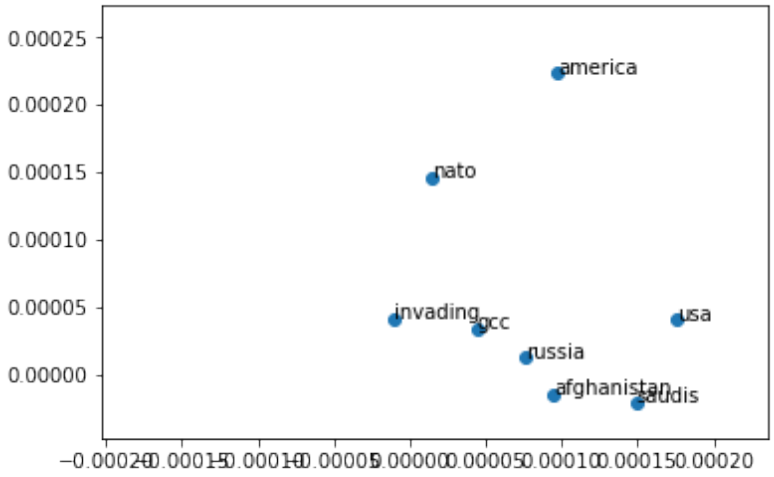


Figure 4.3: USA similar words in pro-ISIS tweets.

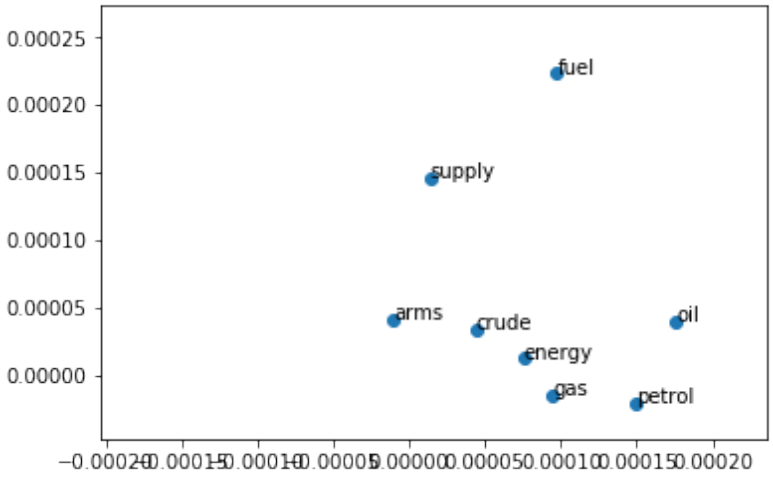


Figure 4.4: Oil similar words in pro-ISIS tweets.

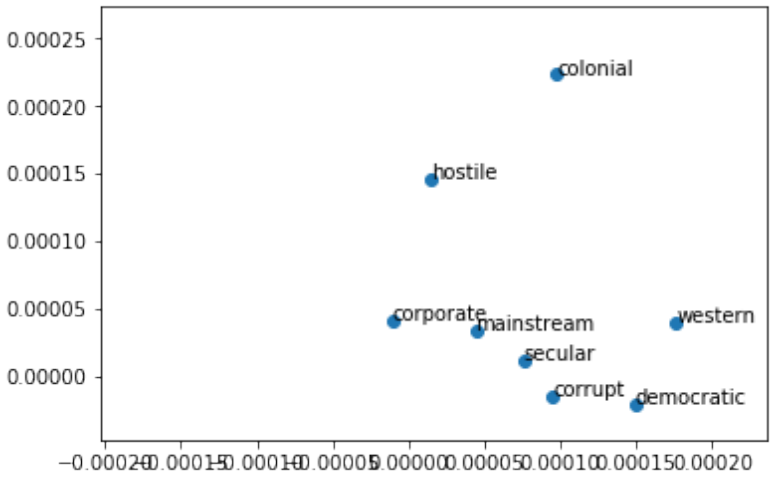


Figure 4.5: Western similar words in pro-ISIS tweets.

Those findings demonstrate the different interpretations of language and beliefs of people according to certain positions, in this case pro-ISIS or anti-ISIS. This means that using this kind of approaches to classify between those groups makes sense as statistical distributions of both groups are different. Additionally, it serves as a kind of verification of the state of the dataset, as the shown results can be intuitively reasoned given the problem context.

4.4 Experiments and results

4.4.1 Introduction

The experiments carried out for the classification task were based on developing the models described in sections 2.8.5 and 2.8.6, which were Recurrent Neural Networks and their combination with Convolutional features. In parallel to those models, simpler linear models were also used as a baseline and some modifications were also made regarding the Neural Network models. The used RNN architecture is shown in Figure 4.6.

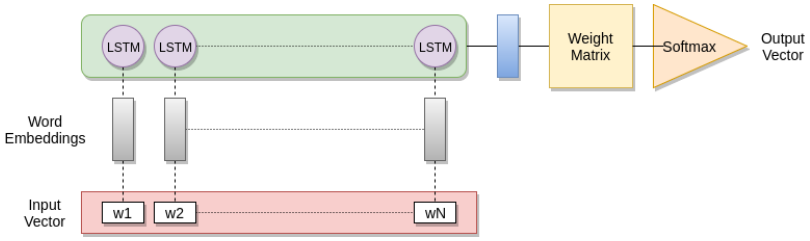


Figure 4.6: RNN architecture.

The initial idea was to create a tweet-level classification model, that is to say, considering individual tweets for performing the classification task. However, as explained in the following sections, another approach was also made by making groups of N-tweets.

The procedure followed in both sections was first exploring the simplest models with the two classes of word embeddings seen in the previous section: pre-trained and self-generated. Afterwards, an improvement over the Recurrent Neural Networks was made including an Attention Layer over them, and finally a hierarchical model was applied to the N-tweet level model, which reached the best metrics.

4.4.2 Tweet-level model

The first experiments were carried out following the previously mentioned methodology, firstly with reduced subsets obtained from all the available data as the training time of each run could last more than one hour in average. Reducing the amount of data speeds up the process but also reduces the F-Score of the models, so it was done only for obtaining some

quick intuition about which model could work better. Once the ideas which performed worst were discarded, the rest of models were evaluated using the whole dataset. The final dataset used for the first approach was split as shown in Table 4.1.

Subset	Num. of instances	Num. of pro	Num. of anti
Train	1,084,520	542,260	542,260
Test	120,500	60,250	60,250
Total	1,205,020	602,510	602,510

Table 4.1: Dataset split.

Regarding Neural Network models, several options were explored. First, the explained RNN architecture was proven using LSTM and GRU cells. Those two types of cells seemed to perform in a similar way, but slightly better in the case of LSTM and though for avoiding duplicate experiments, LSTM cells were chosen. Additionally, a more complex model was also tried with the aim of combining Convolutional Neural Networks for feature extraction and RNNs fed with the CNNs outputs. The results of this combination of topologies were worse than only using the RNN model.

Another interesting fact was using two classes of word embeddings, ones pre-trained and others self-generated from the available corpus. The pre-trained ones which best worked were the Glove embeddings trained on 42 billion documents and containing 300 length vectors for each word. However, results show that embeddings obtained generated from the available corpus performed better than the pre-trained ones.

Those first experiments showed that Recurrent Neural Networks could be useful for this task, but not as powerful as expected as other models could perform almost as well as them. For example, model called NB-SVM [54] was developed and used as a linear baseline. NB-SVM is simple Support Vector Machines variant using Naive Bayes log-count ratios as feature values. Its results were quite similar to the ones obtained with RNNs, althought a bit lower than the one trained with the self-generated word embeddings.

After using the mentioned models, a modification was introduced to the previous RNN architecture using an attention [55] layer which was described before, which tries to focus on specific parts of the text which are more relevant than others. Results were quite interesting, as they improved results using both Glove and self-generated word embeddings but had much more effect using the self-generated ones.

Table 4.2 summarizes all the described experiments.

Model	F-Score
NBSVM (linear baseline)	0.7534
RNN Glove embeddings	0.7486
RNN Self-generated embeddings	0.7915
CNN + RNN Glove embeddings	0.7306
CNN + RNN Self-generated embeddings	0.7328
RNN Glove embeddings + Attention	0.7598
RNN Self-generated embeddings + Attention	**0.8366**

Table 4.2: Tweet-level results.

From the previous table a clear conclusion could be extracted: the linear model performed as well as the Neural Network models except for the ones trained with the self-generated embeddings, and the ones which incorporated the attention layer. At first glance, this could be discouraging as Neural Network models should perform better having such a big dataset available. However, having a glance over some tweets, it was understandable that the models did not perform as good as desired due to a clear fact: users tagged as pro-ISIS did not only post content related to their support to jihadist organizations. The following section shows another approach for dealing with said situation.

4.4.3 N-Tweet-level model

The reason of focusing the classifier at tweet-level was due to the scarce amount of users (566 of each class), but now the problem was that not all tweets from each radical user could be tagged as pro-ISIS. At this point, the best solution was creating a hybrid approach in which instead of analysing tweets individually, they were grouped in small groups and tagged according to the class of their user. That way, the probability of finding a pro-ISIS tweet in a group of N tweets belonging to a pro-ISIS user was much higher and so the classifier could learn better. On the other hand, the dataset was reduced as each instance is compound of N tweets instead of a single tweet.

The question now was choosing the optimal value of N. The first experiments carried out showed that the assumption of grouping tweets clearly improved results. Three different windows were used: 3,5 and 10-tweet level. Results improved as long as the window increased, but there were also two disadvantages for choosing the highest N as possible:

- The available dataset was reduced, and so a worse generalization would be achieved.

- If the model was required to be working live, time would also be a requirement. That way, the time to make a prediction would also depend on the time of gathering five new tweets from a user. That supposes that there is a trade-off between F-Score and time to predict.

Having those reasons in mind, the final windows size was chosen to be 5. Table 4.3 shows the new dataset numbers. It is important to notice that all groups of tweets contained tweets from a single user, being some tweets per user discarded if they did not complete a five-tweet group.

Subset	Num. of instances	Num. of pro	Num. of anti
Train	216,594	108,297	108,297
Test	24,066	12,033	12,033
Total	240,660	120,330	120,330

Table 4.3: 5-tweet level dataset split.

With this new dataset, the experiments were similar to the ones carried out at tweet-level, excepting for the discarded option of using CNN configurations and also adding new topologies which were not explored. The following table shows the RNN results applied to the 5-tweet level corpus. As supposed, results were clearly improved but something curious happened. In this case Glove embeddings performed crearly better than self-generated embeddings without using attention. However, with the attention layer, self-generated embeddings results were clearly better as shown in Table 4.4.

Model	F-Score
RNN Glove embeddings	0.8310
RNN Self-generated embeddings	0.7599
RNN Glove embeddings + Attention	0.8777
RNN Self-generated embeddings + Attention	0.9162

Table 4.4: 5-Tweet-level results.

In this case, the best F-Score was reached using another neural network architecture which takes into account the new data structure of tweet grouping. It is called Hierarchical Attention Network [56] and in this case, the topology consists of a two-level recurrent neural network:

- The first level acts over words, as the previously defined topology. However, the whole five tweets are not fed at once, but in a tweet by tweet form. Additionally, it also contains an attention layer on top of it, and as a result it outputs one number per sentence for the next level of the neural network, which will take it as input.
- The second layer is a sentence encoder (in this case, tweet encoder) which takes as input the result of the previous layer over each tweet. In this case, there will be 5 inputs at this layer, each one obtained from the word-level layer of the 5 analyzed tweets at each time.

This neural network architecture has also been trained with both Glove and self-generated embeddings of the 5-tweet-level classifier, obtaining the best result as shown in table 4.5.

Model	F-Score
Hierarchical attention Glove embeddings	0.8932
Hierarchical attention Self-generated embeddings	**0.9312**

Table 4.5: Hierarchical Attention Network 5-tweet-level results.

Figure 5.6 shows the used architecture of the Hierarchical Attention Network as stated in [56] and its Keras implementation can be found at Appendix 3.

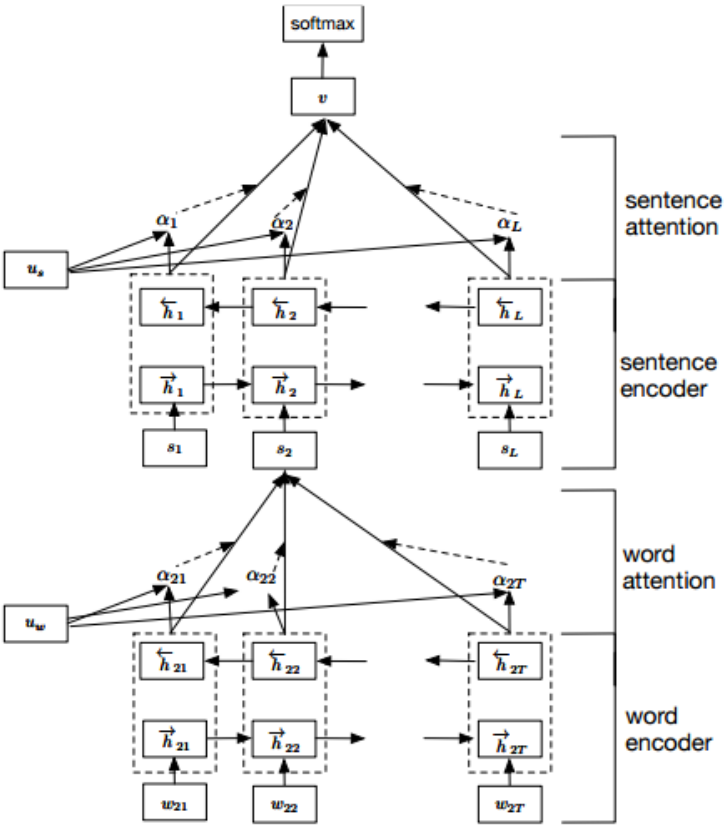


Figure 4.7: Hierarchical attention network architecture [56] .

CHAPTER 5

Conclusions

This chapter will state the achieved goals, some final thoughts about the development of the project and possible future lines of work.

5.1 Achieved Goals

The development of the project has successfully reached the proposed goals. Firstly, the data monitoring system was developed making use of GSI Crawler framework. Data ingestion from diverse Internet sources where radicalism appears was carried out through Web and PDF Scraping techniques. Those scrapers provided a Linked Data output which was enriched by analysis operations performed by Senpy Plugins such as Entity Recognition, Geolocation and Sentiment Analysis.

Once all the analysis were performed, the extracted information was persisted in ElasticSearch, and a visualization layer was built on top of it in a dashboard form. This dashboard shows in real time the data sources that have been scraped along with their analysis. Additionally, the system was given annotation capabilities for later processes, being narrative annotation a use case seeking their automatic detection afterwards.

Lastly, a radicalism tweet classifier was developed following Natural Language Processing principles and making use of Deep Learning techniques. Models were trained using Keras over TensorFlow.

5.2 Final thoughts

The Internet contains huge amounts of information which is really hard to treat. Web pages offer information that can be publicly accessed, but processing it from its raw state is a hard task for humans. The developed system is able to automate tough tasks for humans such as data collection or Data Analysis, accelerating it and being capable of working continuously. However, apart from the quantity of data available at the Internet, another problem is faced when developing these kinds of systems: data is unstructured. Each data source has its own way of displaying data, and the task of structuring all data sources has also been one of the goals of the project. This is specially useful when the extracted information along with the performed analysis are stored for being reused in the future. For accomplishing this objective, Linked Data principles have been useful specially for organizing the diverse sources of information in a proper way aiming at making them available using semantic queries.

Regarding the classification process, the achieved performance was satisfactory. Considering tweets individually results were not as high as desired because each tweet was labeled as radical according to its user, but not all radical user tweets are oriented to propaganda aims. As classifying users was not a feasible task due to the small number of users present at the dataset, an intermediate solution was chosen: analyzing groups of tweets and determining whether they belong to a radical user or not. That way, the probability of a group

of radical tweets containing a radical tweet was much higher than the first approach, but there were not as few examples as in the case of only focusing on users.

It was proven that recurrent neural networks performed well for detecting radicalism in tweets, and could be improved using an attention technique which focuses on the most relevant parts of each tweet. Additionally, it is also remarkable the importance of using the adequate word embeddings as input of the neural network model. The best model was achieved using a hierarchical attention network, which first performed a word-level analysis over individual tweets and then acted at tweet level considering tweets belonging to each group.

5.2.1 Future work

Finally, here are some possibilities of improving the developed work in the future:

- Improving the visualization module with React instead of Polymer Web Components.
- Thanks to the annotation capabilities of the data monitoring platform it is possible to create any corpus which could be used for building future machine learning models. For example, following the Trivalent project, it could be possible to create a narrative detection classifier for knowing if a tweet contains a specific jihadist narrative.
- Using language generation techniques also following machine learning principles, another interesting field could be automating the creation of counter narratives based on the studied narratives.
- For improving the classification model, a recursive neural network could be used instead of a recurrent neural network. Recursive neural networks operate over trees, and they imply the reformatting of the text inputs into a binary tree form, which is not easily achievable.

APPENDIX A

Ethical, Economical, Social and Environmental impact.

A.1 Introduction

This appendix will show the Ethical, Economical, Social and Environmental impacts of this project. As it has been described in the document, the project is mainly centered in the extraction and processing of information from the Internet for radicalism detection and so the realization of this impact analysis makes sense specially concerning ethical and social impacts.

A.2 Ethical impact

This project has clear ethical implications as it gathers data from public profiles on the Internet and tries to classify their tweets according to whether they contain radical oriented content or not. The automatic classification of public profiles has several ethical issues depending on the aim of the process, and in this case the goal of the project is related with public safety of citizens. Trivalent project's goals consist on minimizing the impact of jihadist radicalism through counter-narrative means, and the result of this project is directly related to it. Additionally, security forces of several countries are also partners of

this project, and so the analyzed information can be directly sent to them, who could later do a human interpretation of it.

Considering the new General Data Protection Regulation, anonymous data is not under its scope, and the tweets used for creating the model had been previously anonymised. Additionally, the tweets and users belonging to the radical class were also deleted from Twitter's public feed.

A.3 Economical impact

Regarding the economical impact of the project, following Trivalent project's goal of defining counter-narrative measures for online radicalism, it can be considered that this solution of facing radicalism could be cheaper than other means which require human presence. The focus of this perspective is a preventive model, which is generally higher than reactive models. It is true that this fact is arguable as its efficiency hasn't been proven yet, but having in mind the latest episodes, a preventive approach is required.

Having assumed that this preventive approach is needed, the developed system is cost effective against other methodologies. The cost of manually doing all the tasks performed by the system would be extremely high with respect to this automated approach. A huge number of human labour would be needed, which would suppose a much higher cost in salaries.

A.4 Social impact

The social impact of this project could be intimately linked to the ethic aspects treated before. The goal is to monitor public opinions on social networks, but once again, the ultimate goal is assuring public safety by avoiding terrorist attacks. Additionally, the human labour of security forces of several countries also filter the obtained information from the system, making it even more reliable.

A.5 Environmental impact

This project does not have a clear environmental impact, except for the computing power needed for having the described systems running. The most common way to carry out those experiments is using a shared pool of resources, and so the host machines are being use for multiple applications at the same time. If those resources are allocated on a big cloud service, the whole system would be elastic with respect to demand, and so power consumption could be optimised in a way that only the needed machines would be working

at any time. It is true that the power cost of a whole data center is really intensive, but the needed power to execute all the services in independent machines would be several times higher.

APPENDIX B

Economic budget.

B.1 Introduction

This appendix describes the economic budget regarding the development of the project, considering material and human resources.

B.2 Material resources

This project requires a high quality computer for its development and a server for its deployment. If the server has additional computing resources, deep learning models can be trained there and so the requisites of the computer could be relaxed.

The most resource consuming process was the training of the word embeddings model and of the neural network training. We estimate the need of 16 GB of RAM and optionally a GPU for accelerating the model training process. A computer of around 1,000 euros would be sufficient for this task, and the GPU could even double this budget.

Regarding the deployment server for the service, there isn't a exhaustive need of resources and so a common server could be used, being its price from 1,000 euros on. This server doesn't need to be fully dedicated to the deployed service, and even a contracted VPS service solution could be used. The diversity of VPS vendors is really high, but an average reasonable price could be 20 euros per month.

B.3 Human resources

For calculating the budget dedicated for the development of the project, we suppose a software engineer for developing the whole solution in a full time basis, estimating four months of work. Supposing a gross monthly salary of 1,500 euros, the total cost would be 6,000 euros.

There would also be needed another engineer executing tasks related with the system deployment, availability and security. This person should be full time hired with the purpose of fixing any incidence happening at any time, but as this would not happen frequently, he could also be focused on similar tasks for other projects. Its associated cost would also be around 1,500 euros each month the system is up.

APPENDIX C

Keras implementation of Hierarchical Attention Network

Listing C.1: Hierarchical Attention Network implementation in Keras

```
WORD_NUM = MAX_LEN
SENT_NUM = N
max_features = len(word_dict) + 1
batch_size = 64

num_lstm = 64
num_dense = 64
rate_drop_lstm = 0.35
rate_drop_dense = 0.35




embedding_layer = Embedding(len(word_dict) + 1,
                            EMBEDDING_DIM,
                            weights=[embedding_matrix],
```

```
                            input_length=WORD_NUM,
                            trainable=True)

sentence_input = Input(shape=(WORD_NUM,),dtype='int32')
embedded_sequences = embedding_layer(sentence_input)

activations = Bidirectional(LSTM(num_lstm, dropout=rate_drop_dense,
    recurrent_dropout=rate_drop_dense,return_sequences=True))(
    embedded_sequences)

attention = TimeDistributed(Dense(1, activation='tanh'))(activations)
attention = Flatten()(attention)
attention = Activation('softmax')(attention)
attention = RepeatVector(2*num_lstm)(attention)
attention = Permute([2, 1])(attention)

sent_representation = merge([activations, attention], mode='mul')
sent_representation = Lambda(lambda xin: K.sum(xin, axis=-1))(
    sent_representation)

sentEncoder = Model(inputs=sentence_input,outputs= sent_representation)


text_input = Input(shape=(SENT_NUM, WORD_NUM), dtype='int32')
text_encoder = TimeDistributed(sentEncoder)(text_input)

activations_sent = Bidirectional(LSTM(num_lstm, dropout=rate_drop_dense,
    recurrent_dropout=rate_drop_dense,return_sequences=True))(text_encoder)

attention_sent = Dense(1, activation='tanh')(activations_sent)
attention_sent = Flatten()(attention_sent)
attention_sent = Activation('softmax')(attention_sent)
attention_sent = RepeatVector(2*num_lstm)(attention_sent)
attention_sent = Permute([2, 1])(attention_sent)

text_representation = merge([activations_sent, attention_sent], mode='mul')
text_representation = Lambda(lambda xin: K.sum(xin, axis=1))(
    text_representation)



preds = Dense(1, activation='sigmoid')(text_representation)

model = Model(inputs=text_input, outputs=preds)
```

```
model.compile(loss='binary_crossentropy',
        optimizer='adam',
        metrics=['accuracy'])
```

Bibliography

[1] Expert system cogito. http://www.expertsystem.com/es/productos/cogito/.

[2] Google maps geocoding. https://developers.google.com/maps/documentation/geocoding/start.

[3] Gsi crawler documentation. http://gsicrawler.readthedocs.io/en/latest/.

[4] Hypothes.is. https://web.hypothes.is/.

[5] Introduction to neural networks. https://gormanalysis.com/introduction-to-neural-networks/.

[6] Keras documentation. https://keras.io/.

[7] Pdf miner. https://github.com/euske/pdfminer.

[8] Pdf query. https://github.com/jcushman/pdfquery.

[9] Trivalent project. http://trivalent-project.eu/.

[10] Understanding cnn networks for nlp. http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/.

[11] Understanding gru networks. https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be.

[12] Understanding lstms. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[13] ¿cómo se radicaliza un 'millennial' musulmán de ripoll? https://www.elconfidencial.com/espana/cataluna/2017-08-19/radicalizacion-de-ninos-y-adolescentes_1430994/.

[14] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[15] Ben Adida, Mark Birbeck, Shane McCarron, and Steven Pemberton. Rdfa in xhtml: Syntax and processing. *Recommendation, W3C*, 7, 2008.

[16] Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65, 2003.

[17] Oscar Araque, Marco Guerini, Carlo Strapparava, and Carlos A Iglesias. Neural domain adaptation of sentiment lexicons. In *Affective Computing and Intelligent Interaction Workshops and Demos (ACIIW), 2017 Seventh International Conference on*, pages 105–110. IEEE, 2017.

[18] Khalid Belhajjame, James Cheney, David Corsar, Daniel Garijo, Stian Soiland-Reyes, Stephan Zednik, and Jun Zhao. Prov-o: The prov ontology. *W3C Working Draft*, 2012.

[19] Steven Bird and Edward Loper. Nltk: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics, 2004.

[20] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far. *International journal on semantic web and information systems*, 5(3):1–22, 2009.

[21] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[22] Dan Brickley and Libby Miller. Foaf vocabulary specification 0.91, 2007.

[23] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[24] James Clark, Steve DeRose, et al. Xml path language (xpath) version 1.0, 1999.

[25] World Wide Web Consortium et al. Json-ld 1.0: a json-based serialization for linked data. 2014.

[26] Kris De Volder. Jquery: A generic code browser with a declarative configuration language. In *International Symposium on Practical Aspects of Declarative Languages*, pages 88–102. Springer, 2006.

[27] M Erdmann, B Fischer, R Fischer, and M Rieger. Design and execution of make-like, distributed analyses based on spotify's pipelining package luigi. In *Journal of Physics: Conference Series*, volume 898, page 072047. IOP Publishing, 2017.

[28] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[29] Yoav Goldberg and Omer Levy. word2vec explained: Deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[30] Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine.* " O'Reilly Media, Inc.", 2015.

[31] Ramanathan V Guha, Dan Brickley, and Steve Macbeth. Schema. org: evolution of structured data on the web. *Communications of the ACM*, 59(2):44–51, 2016.

[32] Peter Haase, Jeen Broekstra, Andreas Eberhart, and Raphael Volz. A comparison of rdf query languages. In *International Semantic Web Conference*, pages 502–517. Springer, 2004.

[33] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[34] Sebastian Hellmann, Jens Lehmann, Sören Auer, and Martin Brümmer. Integrating nlp using linked data. In *International semantic web conference*, pages 98–113. Springer, 2013.

[35] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[36] Graham Klyne and Jeremy J Carroll. Resource description framework (rdf): Concepts and abstract syntax. 2006.

[37] D Krech et al. Rdflib python library. Technical report, Tech. rep, 2002.

[38] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[39] Wes McKinney. *Python for data analysis: Data wrangling with Pandas, NumPy, and IPython.* " O'Reilly Media, Inc.", 2012.

[40] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[41] Alistair Miles and Sean Bechhofer. Skos simple knowledge organization system reference. *W3C recommendation*, 18:W3C, 2009.

[42] Ryan Mitchell. *Web scraping with Python: collecting data from the modern web.* " O'Reilly Media, Inc.", 2015.

[43] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[44] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[45] Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[46] Eric Prud, Andy Seaborne, et al. Sparql query language for rdf. 2006.

[47] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[48] J Fernando Sánchez-Rada, Carlos A Iglesias, Ignacio Corcuera, and Oscar Araque. Senpy: A pragmatic linked sentiment analysis framework. In *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on*, pages 735–742. IEEE, 2016.

[49] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[50] A Seaborne. Fuseki: serving rdf data over http, 2011.

[51] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[52] Joseph Turian, Lev Ratinov, and Yoshua Bengio. Word representations: a simple and general method for semi-supervised learning. In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics, 2010.

[53] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[54] Sida Wang and Christopher D Manning. Baselines and bigrams: Simple, good sentiment and topic classification. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 90–94. Association for Computational Linguistics, 2012.

[55] Yequan Wang, Minlie Huang, Li Zhao, et al. Attention-based lstm for aspect-level sentiment classification. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 606–615, 2016.

[56] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, 2016.