TÉCNICO LISBOA

# ConnectionLens: Entity and Relationship Extraction from French textual data sources

**Catarina Pinto Conceição**

Thesis to obtain the Master of Science Degree in

## Information Systems and Computer Engineering

Supervisors: Prof. Helena Isabel de Jesus Galhardas
Dr. Ioana Gabriela Manolescu-Goujot

### Examination Committee

Chairperson: Prof. José Carlos Martins Delgado
Supervisor: Prof. Helena Isabel de Jesus Galhardas
Member of the Committee: Prof. Bruno Emanuel Da Graça Martins

**October 2020**

# Abstract

As a result of the large amounts of data digitally available nowadays, journalists are turning their attention to data processing and visualization, a task called *data journalism*. In investigative journalism, the available data is used to find connections between entities and analyze their nature. CONNECTIONLENS is a software prototype that addresses the investigative journalism's issues of having data from different sources and different formats, while allowing keyword-based queries to find connections. To obtain the entities and connections in textual data sources it is necessary to perform Named-Entity Recognition (NER) and Relationship Extraction (RE). We propose to develop a solution for NER and RE for French news texts that can be incorporated in CONNECTIONLENS. Our goal is to adapt and make use of tools, more specifically, third-party libraries, for both NER and RE, to create machine learning models capable of extracting named-entities and relationships, respectively, from French texts. In addition, to provide a comprehensive evaluation of these models using precision, recall and $F1$-score for NER, and precision-recall curves, area under the curve (AUC), micro-$F1$ and Precision@N for RE. Finally, to select the best performing model for each task, to be integrated in CONNECTIONLENS. The best performing model for NER achieved an overall $F1$-score of $73.31\%$. And, the best performing model for RE achieved an AUC and a micro-$F1$ of $97.10\%$ and $91.78\%$, respectively.

# Keywords

Information Extraction, Natural Language Processing, Named-Entity Recognition, Relationship Extraction, Deep Learning, Distant Supervision

# Resumo

Como resultado da grande quantidade de dados disponíveis digitalmente hoje em dia, os jornalistas estão a mudar o seu foco para processamento e visualização de dados, numa tarefa denominada *jornalismo de dados*. No jornalismo de investigação, os dados disponíveis são usados para descobrir conexões entre entidades e analisar a natureza das mesmas. CONNECTIONLENS é um protótipo de *software* que tenta resolver os problemas do jornalismo de investigação de ter dados de diferentes fontes e com diferentes formatos, permitindo também efetuar *queries* baseadas em palavras-chave para encontrar conexões. Para obter entidades e conexões em fontes de dados textuais é necessário realizar Reconhecimento de Entidades Mencionadas (REM) e Extração de Relações (ER). Nós propomos o desenvolvimento de uma solução para REM e ER para textos de notícias em Francês que possa ser integrada no CONNECTIONLENS. O nosso objetivo é adaptar e usar ferramentas, mais especificamente bibliotecas, tanto para REM e RE, para criar modelos de aprendizagem automática capazes de extrair entidades mencionadas e relações, respetivamente, de textos franceses. Adicionalmente, efetuar uma avaliação extensiva desses modelos, usando precisão, abrangência e medida $F1$ para REM, e curvas de precisão-abrangência, área sob a curva (AUC), *micro*-$F1$ e Precisão@N para ER. Finalmente, selecionar o modelo com melhor desempenho para cada tarefa, para serem integrados no CONNECTIONLENS. O modelo com melhor desempenho para REM obteve uma medida $F1$ global de $73.31\%$, e o modelo com melhor desempenho para ER obteve uma AUC e uma micro-$F1$ de $97.10\%$ e $91.78\%$, respetivamente.

# Palavras Chave

Extração de Informação, Processamento de Língua Natural, Reconhecimento de Entidades Mencionadas, Extração de Relações, Aprendizagem Profunda, Supervisão Distante

# Contents

# List of Figures

# List of Tables

# 1

# Introduction

## Contents

Traditionally, journalists focused on gathering data and being the first to deliver news. Nowadays, with the large amount of data digitally available, journalists are shifting their focus to extracting, transforming and visualizing data with the goal of having a good story to tell. These data processing and visualization tasks are typically known under the name *data journalism* [1]. When handling large amounts of data, it is crucial to choose what is of interest and be able to interpret data, thus avoiding to present raw data. Furthermore, investigative journalism uses available data and tries to find connections between subjects, objects, people (i.e., entities) and analyze their nature. There is typically a network of interconnections between those entities that is not visible. It is the intention of investigative journalism to bring to light these interconnections, through the analysis of combined data from different data sources.

The set of data sources used in investigative journalism may be heterogeneous and independently produced. Moreover, it is necessary to deal with the changing nature of the data sources because journalists are always collecting more data with different structure and format. Once the data is integrated, journalists need to query it to find connections. They do not know the exact structure of the data, so queries are typically keyword-based.

CONNECTIONLENS [2] is a software prototype that was developed to address the investigative journalism problem described above. This prototype supports keyword search across a set of heterogeneous and independently produced data sources, to find connections. It deals with different types of data sources, namely JSON documents, text files, RDF graphs and relational tables. CONNECTIONLENS was developed in the context of the ContentCheck ANR project[1] with the French newspaper Le Monde, so it focuses on French data. This project was a collaborative project between the Inria CEDAR team[2] and AIST Japan[3]. Example 1 shows an example of a query that can be answered using CONNECTIONLENS. The example was raised by Le Monde and taken from [2].

> **Example 1** In the 2007 National Elections in France, there was a large amount of first-time National Assembly elected members. Naturally, journalists tried to find as much data as possible about them. More specifically, they asked "what connections exist between the elected representatives and companies?". Therefore, they sought to identify direct financial interests (which must be disclosed by the members), but also indirect connections which might generate conflicts of interest, e.g., being a close collaborator of a company's CEO (which must be discovered by the press).

Taking into consideration Example 1, when posing the following query to CONNECTIONLENS : "En Marche companies", connections between elected representatives from the political party "En Marche" and companies will be found. In particular, Figure 1.1 shows a collection of data sources over which CONNECTIONLENS will provide an answer to the query, in particular: ($i$) a JSON data source DS1 with

[1] https://team.inria.fr/cedar/contentcheck/
[2] https://team.inria.fr/cedar/
[3] https://www.aist.go.jp/waterfront/

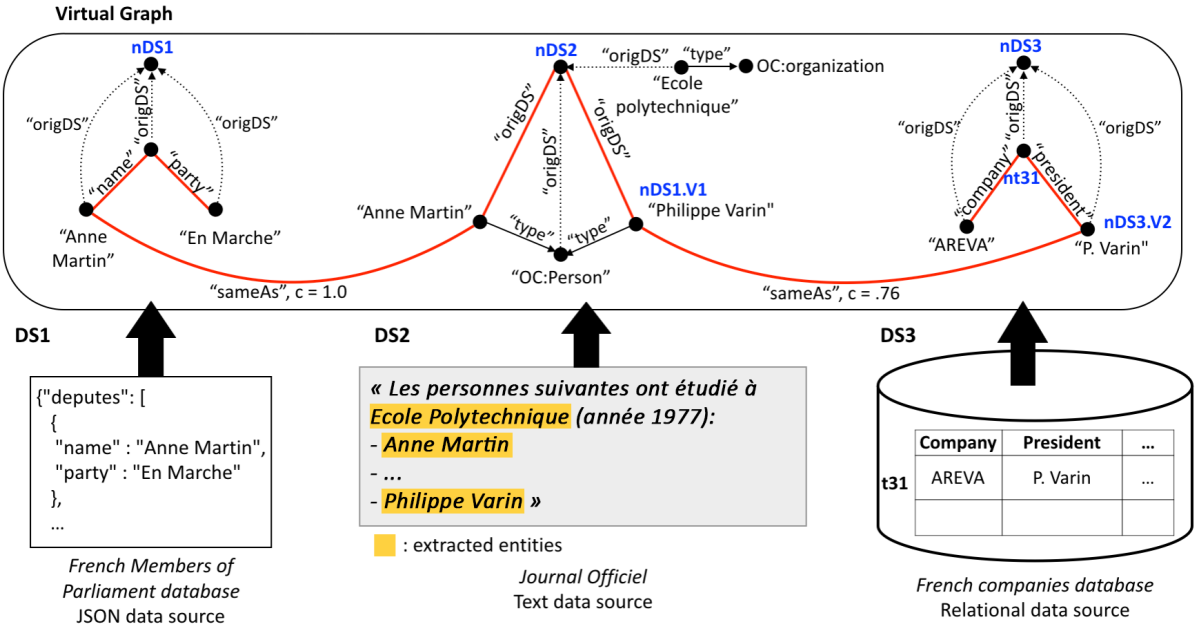


**Figure 1.1:** CONNECTIONLENS: Data source collection, virtual graph and a possible answer tree

National Assembly representatives, $(ii)$ a text data source DS2 from Journal Officiel that contains a list of the alumni of Ecole Polytechnique (where several French company executives studied), and $(iii)$ a relational data source DS3 of companies and their CEOs.

In CONNECTIONLENS all data sources are mapped into a single virtual graph, as it is represented in Figure 1.1. Most nodes and edges of the virtual graph come from the data sources. All data sources, except for the text type, have inherently defined entities and connections, so, defining nodes and edges is only necessary to map the inherent nodes and connections to the virtual graph. Edges in the virtual graph can also be links between two nodes (of the same or from different data sources) whose data is considered to be similar. These are called *sameAs* links e.g., "Philippe Varin" and "P. Varin".

The answer to a keyword-based query posed to the virtual graph is given in the form of an answer tree, which is part of the virtual graph. Several answer trees may be returned for a query, each with an associated score. One possible answer tree for the query "En Marche companies" is represented in Figure 1.1 by a red line. Each keyword in the query matches a node or an edge in this answer tree. The answer tree in Figure 1.1 shows a connection between "Anne Martin", a representative from "En Marche" and the company "AREVA". In fact, in DS1 there is a node representing "Anne Martin" as being a member of the party "En Marche". This node matches a node "Anne Martin" from DS2, connected by a "sameAs" link, because the two strings are similar. The occurrence of "Anne Martin" in DS2 means that she studied in Ecole Polytechnique. "Philippe Varin" also studied in Ecole Polytechnique, as it is represented by a node from DS2. The company "AREVA" is represented by a node in DS3, whose president is "P. Varin" also represented by a node in DS3. This node connects to the node "Philippe

Varin" in DS2 by a "sameAs" link because the strings are similar.

A textual data source, such as the Journal Officiel (DS2) in Figure 1.1, is a particularly interesting data source to integrate. The text is written in Natural Language (French, in this case), that is unstructured, so it has no pre-defined format or model, unlike a relational database. To obtain the entities and connections present in a natural language text, it is necessary to perform Information Extraction (IE) [3], in particular, Named-Entity Recognition and Relationship Extraction.

Named-Entity Recognition (NER) is a typical task of IE that focuses on identifying names of entities and then classifying them into a pre-defined set of classes, like people, locations, organizations and others. For example, in Figure 1.1, consider that NER was performed over DS2 with the goal of identifying names of entities of the type *Organization* and *Person*. "Anne Martin" and "Philippe Varin" would be detected and classified as *Person* and "Ecole Polytechnique" as *Organization*.

Relationship Extraction (RE) is a task of IE that aims at extracting relationships, usually from a pre-defined set, between the previously identified entities. In Figure 1.1, RE was performed over DS2, with the goal of identifying the relationship *StudiedIn*. Given the previously recognized entities, the relationships *(Anne Martin, Ecole Polytechnique, StudiedIn)* and *(Philippe Varin, Ecole Polytechnique, StudiedIn)* would be detected.

The goal of this thesis is to develop a solution for NER and RE for French news texts that can be incorporated in CONNECTIONLENS.

## 1.1 Existing NER and RE Solutions

Approaches for performing both NER and RE may consist in using a software tool or implementing a technique from scratch. Software tools implement techniques and are able to perform the given task *off-the-shelf* and/or facilitate its implementation.

Regarding NER, the considered standard techniques are: $(i)$ **supervised machine learning** and $(ii)$ **rule-based**. NER is seen as a sequence labeling problem where the goal is to assign a label, i.e., a named-entity type, to each word in an input sequence of words, e.g., a sentence. Rule-based methods are easy to implement and interpret, and useful when extracting something with a regular format e.g. extracting phone numbers. Nevertheless, linguistic knowledge and a lot of work is required to create a rule-based system. Moreover, supervised methods require an algorithm that will learn from a considerable amount of labeled data, preferably from the same domain as the data that is going to be labeled, i.e., news. These methods can be further divided in **feature-based** and **neural-based**, and the latter, currently achieves state-of-the-art results.

The software tools that are available for NER can be: $(i)$ **black-box** or $(ii)$ **third-party libraries**. Black-box software tools are usually offered as a web service made available by an API. We make a

request to the service sending text as input and it returns the named-entities it was able to extract from it. We typically have no knowledge of what is happening internally in the system: technique, and possibly the training data, are usually unknown. Moreover, these solutions usually support a maximum number of accesses per time period (for example, per day). There are several black-box web services capable of dealing with French, e.g., Open Calais[4], IBM Watson NLU[5], however the constraints they present are the ones described above and this is not desired when integrating into CONNECTIONLENS.

On the other hand, third-party libraries implement a technique and only require train data to be given as input to obtain a machine learning model. They also make pre-trained models available, which are machine learning models already trained and parameterized. With pre-trained models, the technique is known, however, there may be inconsistencies on the domain level between the data used to train the pre-trained models and the data subsequently used as input.

We found several third-party libraries for NER that implement state-of-the-art supervised neural-based techniques. In addition there are several freely available datasets in French annotated with named-entities that can be given as input to those tools to create a French NER model. Some of the third-party libraries we found, able to perform NER, also make available French pre-trained models, in particular, Flair [4] and SpaCy[6].

In what concerns RE, there are no datasets available, in French, with annotated relationships between entities. Therefore, we are limited to techniques that do not require manually labeled data. Again, rule-based methods require a high amount of linguistic, in this case French, knowledge, therefore the most relevant techniques are: $(i)$ **semi-supervised**, $(ii)$ **distantly supervised** and $(iii)$ **unsupervised** also referred as Open Information Extraction (Open IE).

The software tools we found capable of performing RE can be categorized in the same way as the ones for NER, i.e., **black-box** and **third-party libraries**, with the addition of **Open IE systems**. Open IE systems implement an Open IE technique, for a given language, that is able to extract relationships from text given as input. In these types of systems there is not a defined set of relationships types that can be extracted; the system extracts all types of relationships it is capable of.

We found two tools capable of performing RE for French *off-the-shelf*: IBM Watson NLU and a French implementation of ReVerb [5]. Again, IBM Watson NLU is a black-box web service with constraints and its implementation details are unknown, undesirable traits for a CONNECTIONLENS integration. Moreover, the French adaptation of ReVerb is an Open IE system where there is not a defined set of relationships that can be extracted. This is equally not desired for extracting relationships in CONNECTIONLENS. Regarding third-party libraries, we only found one that allowed training non supervised models, which is OpenNRE [6]. It integrates training of bag-level RE models, a widely applied method for distantly

[4]https://permid.org/onecalaisViewer
[5]https://www.ibm.com/watson/services/natural-language-understanding
[6]https://spacy.io/

supervised RE. Moreover, distantly supervised RE proposes a procedure that automatically creates annotated data for training machine learning models, using relationship instances from a knowledge base (KB), and sentences expressing the relationships, from a text corpus.

## 1.2 Objectives

Our goal is to adapt and make use of tools, more specifically, third-party libraries, for both NER and RE, to create machine learning models capable of extracting named-entities and relationships, respectively, from French texts. In addition, we aim at performing a comprehensive evaluation of these models using precision, recall and $F1$-score for NER, and precision-recall curves, area under the curve, micro-$F1$ and Precision@N for RE. Finally, we will choose the best performing models for each task to be integrated in CONNECTIONLENS.

## 1.3 Contributions

The main contributions of this work are as follows:

1. Pre-processing three different French NER datasets, i.e., Quaero Old Press Extended Named Entity, KB Europeana Newspapers NER and WikiNER, with the goal of uniformizing and combining them to have as much information as possible for training. Additionally, data exploration of each pre-processed dataset was performed, with the intention of learning their characteristics, namely the number of sentences, tokens and named-entities.

2. The creation of multiple French NER machine learning models using third-party libraries that implement neural-based state-of-the-art techniques, namely, Flair and SpaCy, and trained using the pre-processed Quaero dataset, divided in train, development and test sets. The models were evaluated using the familiar metrics used to evaluate NER models: precision, recall and $F1$-score. The best performing model out of all was selected, that achieved an overall $F1$-score of $82.44\%$ on the test set of the pre-processed Quaero dataset.

3. A comprehensive evaluation of the created and selected NER model against existing French pre-trained NER models, and the model previously used in CONNECTIONLENS for French, using the familiar metrics. To impartially evaluate the models we used the FTBNER dataset, which we pre-processed. According to the results, we trained a final model that outperformed all models with an overall $F1$-score of $73.31\%$ on the FTBNER dataset.

4. The creation of a French RE dataset using distant supervision and French DBpedia [7] as the KB and Wikpedia articles as the text corpus. We filtered and grouped the relationships we retrieved

from DBpedia and generated negative relationship instances, that express the non-existence of a relationship. For each relationship instance, we collected all sentences in which the related entities co-occur.

5. The creation of a French RE model using OpenNRE, a third-party library that implements bag-level training, a widely applied method for distantly supervised RE.

6. Evaluation and experiments with dataset variants for training the French RE model, that we evaluated using held-out evaluation, where a part of the relationship instances is "held-out" to create a test set. We used the following metrics to evaluate the models, that resulted from the different variants: precision-recall curves, area under the curve (AUC), micro-$F1$ and precision@N. The best performing model presented an AUC and a micro-$F1$ of $97.10\%$ and $91.78\%$, respectively.

The description of the NER solution, i.e., contributions 1, 2 and 3, is included in the paper [8], accepted to the 36th Conference "Gestion de Données - Principes, Technologies et Applications" (BDA 2020).

## 1.4 Thesis Outline

The rest of the document is organized as follows: Chapter 2 provides background about important IE concepts. Chapter 3 details the related work, in what concerns NER and RE. Chapter 4 describes the approach that was taken regarding NER, in particular, how we created our French NER model. Chapter 5 is dedicated to the RE approach, specifically, we detail how we created a distantly supervised French RE model. Chapter 6 presents the experimental evaluation carried out for both NER and RE, in particular the obtained results, and the integration of the models in CONNECTIONLENS. Finally, Chapter 7 concludes this document by presenting the main conclusions and possible future work.

# 2

# Background on Information Extraction

## Contents

Named-Entity Recognition (NER) and Relationship Extraction (RE) are Information Extraction (IE) tasks. In this section, we introduce the main topics regarding IE.

Data sources that need to be analyzed and integrated can be structured or unstructured. A structured data source has a pre-defined data model, thus the data is highly organized, and usually resides in a relational database. This type of data is easily searchable and processed by machines. Yet, many data sources available in the world are unstructured. Unstructured data sources do not have a defined format and cannot be stored in a relational database. Furthermore, that there is a large amount of this kind of data sources available. Therefore, it is difficult and sometimes impossible to obtain relevant search results. IE [3, 9–11] aims at identifying and classifying data from unstructured data sources in semantic classes. This process can be perceived as a mapping of unstructured data sources to a structured form.

Most of the advances in IE have been in natural language texts. However, unstructured data also includes images, video, audio, social media activity, etc. Since the goal of this thesis is to recognize and extract named-entities and relationships from text, the focus of the rest of this section will be on textual data sources. Extracting information from natural language texts relies and may be seen as a higher task of Natural Language Processing (NLP) which is the process of analyzing and manipulating natural language texts in a way that is understandable by computers.

The extraction of semantic information is possible through the detection of smaller sub-structures in text. Humans are able to discover the meaning of a sentence through the meaning of the sentence constituents, their ordering, dependencies and realization. Texts are therefore somewhat regular in terms of the patterns shown and their organization. Through these regularities, it is possible to extract semantic information even if not "immediately computationally transparent" [9].

IE retrieves small parts of the text that belong to a pre-defined set of semantic classes. These classes can be entities e.g., *Person* or *Organization*, relationships between them e.g., *Located* or *Family*, among others. A pre-defined number of attributes can also be extracted. For example, consider that we are extracting and classifying house divisions from a natural language text. Besides the type of house divisions, we can also extract attributes about the room like the shape, the dimensions, etc. The semantic classes might be defined at different granularity levels depending on the application or need, for example, *Location* or *Country* and *City*.

IE may be performed over a small text excerpt or a sentence, but also over paragraphs and documents. The latter occurs when context is important, and for that reason more than one sentence spanning one or more texts is necessary to perform the desired extraction. Furthermore, IE is typically applied to a particular domain (closed domain) e.g., news, medicine, meaning that the textual information and the semantic structures are related to that domain. However, an ideal IE system should be able to perform independently of the domain (open domain).

## 2.1 Information Extraction Pipeline

A basic IE pipeline [3, 10, 11] is shown in Figure 2.1. A *raw natural language text* is given as input to be processed by the pipeline. Then, IE can be divided in four major tasks: Segmentation, Named-Entity Recognition, Relationship Extraction and Coreference Resolution. The raw natural language text given as input to the pipeline also needs to undergo a *text pre-processing* step. Overall, the various IE tasks and the *text pre-processing* step can use a set of *auxiliary resources*.

The raw natural language text is first segmented into sentences and segmented into words (also called tokenization) in the *Segmentation* task (Task 1). Tokenization divides a text into smaller linguist units, called tokens, which can be words, numbers or punctuation. Tokenization often separates punctuation from words and keeps the punctuation as a token, because besides constituting different units, it may be useful. For example, periods are good indicatives of sentence boundaries. Also, a punctuation mark that occurs internally in a word is important to be kept. Examples include abbreviations (e.g., *P. Varin*), compound words with an hyphen (e.g., *Comédie-Française*), contractions (e.g., *l'Isère*), etc. A tokenizer can also expand contractions e.g., *le Isère* or tokenize more than one word in a single token e.g., *"Anne Martin"* instead of *"Anne"* and *"Martin"*.

Sentence segmentation is performed through the detection of punctuation which indicates sentence boundaries: question marks, exclamation points, periods. The *Segmentation* task is mandatory and essential in the IE pipeline because all other tasks use the tokens that result from this step.

Special care must be taken, as mentioned before, because periods may cause problems of ambiguity because they may be placed at the end of sentences or as abbreviation marks. In fact, the same period character may indicate both the end of a sentence or mark an abbreviation if an abbreviation which ends in a period is at the end of a sentence.

The *tokenized sentences* that are produced by segmentation go through a *text pre-processing* step with the help of *auxiliary resources*, more specifically *NLP pre-processing libraries*. The *text pre-processing* task entails several sub-tasks namely: $(i)$ cleaning e.g., removing noise or special characters, $(ii)$ normalization e.g., transforming words to their root, as well as $(iii)$ linguistic annotation e.g., associating part-of-speech tags to tokens, that will be helpful in the succeeding tasks.

*Cleaned and normalized tokens* will be assigned a named-entity class, from a set of pre-defined types of named-entities (including a type for not-entity), in the *Named-Entity Recognition* task (Task 2). The *linguistic annotations* referred by the *text pre-processing* step and a *structured database* containing known entities may help in the recognition of the entities. Additionally, *labeled unstructured text* may participate in the creation of the NER system and it is also helpful for evaluating its performance. As a result of the *Named-Entity Recognition* task, we have a natural language text with *annotated entities*.

Using the text with *annotated entities*, the *Relationship Extraction* task (Task 3) identifies the relationship between a pair of annotated entities in each sentence. Analogously to the *Named-Entity Recogni-*

*tion* task, *linguistic annotations* resulting from the *text pre-processing* step are also used. A *structured database* of known entities and relationships, and *labeled unstructured text* also helps the RE. This task will output a set of *relationship triples* composed of two entities and the relationship between them.

Finally, the *Coreference Resolution* task (Task 4), takes the *relationship triples* and finds entities that refer to the same entity and collapses them into one e.g., "Emmanuel Macron", "President of France", "Macron" are all collapsed to the same named-entity.

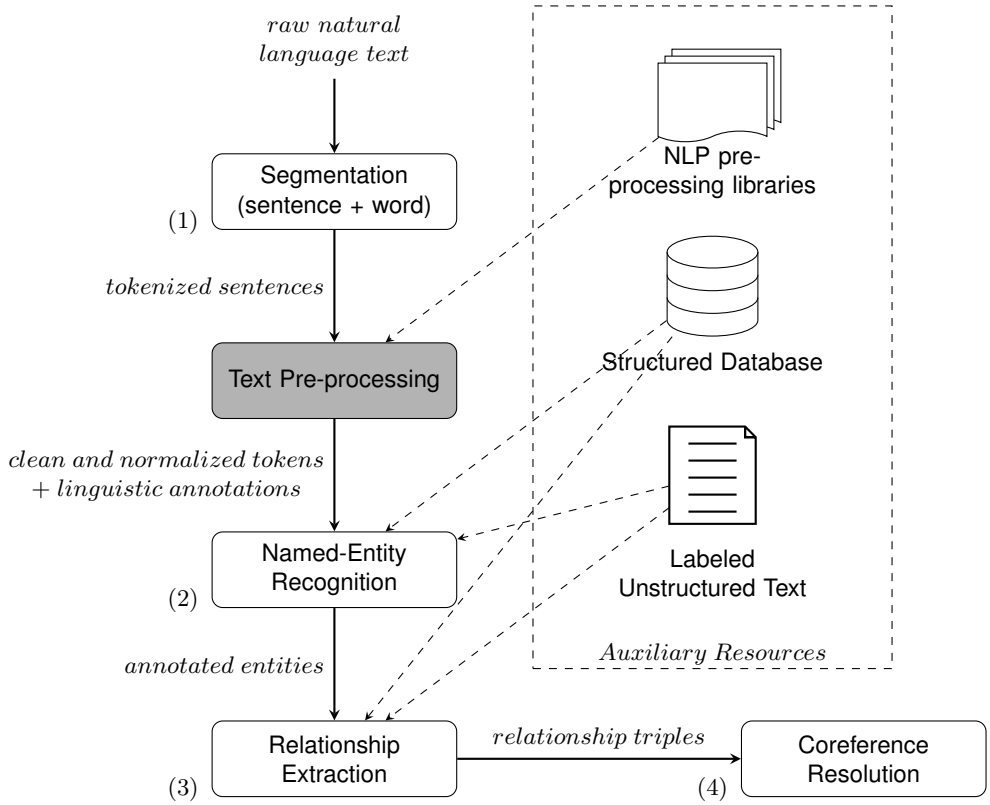


**Figure 2.1:** Information Extraction Pipeline

## 2.2 Auxiliary Resources

Various auxiliary resources can facilitate the extraction task [3], in particular: $(i)$ structured databases, $(ii)$ labeled unstructured text and $(iii)$ NLP pre-processing libraries.

*Structured databases* are populated with known familiar entities and relationships that can help during extraction. *Labeled unstructured text* or a set of labeled unstructured texts (known as corpus), in general manually labeled, are useful for some types of extraction whose creation and initialization depends on them. Moreover, they are used to validate and evaluate the extraction system.

The classification of the extracted information units requires a pre-processing step where each unit

e.g., word, sentence, etc, is transformed into a feature vector. *Features* are properties or attributes that describe the units e.g., for a word: its part-of-speech tag, if it is a capitalized word, etc. The set of all features for some data is called the *feature space*.

*NLP pre-processing libraries* are capable of extracting linguistic information, using natural language processing tools, that will work as features in the extraction task. The following NLP pre-processing libraries are typically used:

1. *Sentence analyzer and Tokenizer:* A sentence analyzer and a tokenizer perform, respectively, sentence segmentation and word segmentation (tokenization). As a result of using both libraries we have sentences and their respective tokens, i.e., tokenized sentences.

2. *Part-of-Speech (POS) tagger:* The tagger assigns a morphosyntactic class (also referred to as grammatical category), from a fixed set, to each word, e.g., noun, verb, adjective, pronoun. A well-known set of POS tags is the *Penn Treebank POS tag set*[1]. The following example, taken from [3], shows a sentence annotated with the *POS tags* of each word, from the *Penn Treebank POS tag set*, is the following:

   The/DT University/NNP of/IN Helsinki/NNP hosts/VBZ ICML/NNP this/DT year/NN,

   where *DT* is a determiner, *NN* is a noun, *NNP* is a proper noun and *IN* is a preposition or subordinating conjunction. The *POS tags* can be used as features during extraction.

3. *Parser:* The parser performs a syntactic analysis of sentences i.e, parsing of sentences. A parser divides sentences in their syntactic constituent *phrases*, e.g., *verb phrase*, *noun phrase*, *adjective phrase*, *prepositional phrase*. A *phrase* is a word or a group of words that has some grammatical meaning in the sentence. Most phrases have a word more important than the others, that is called the head of the phrase, which commonly identifies the type of the phrase e.g., if a phrase has a head that is a noun, that the phrase is a noun phrase.

   A syntactic structure, *parse tree*, is assigned to a sentence as result of parsing. The output parse tree is a dependency structure which can be used in NER, since a named-entity tends to correspond to a noun phrase. It also facilitates the extraction of relationships between entities.

   An example parse tree for the sentence [3]: "Haifa, located 53 miles from Tel Aviv will host ICML in 2010", is shown in Figure 2.2.

   The parse tree shown in figure 2.2 is, specifically, a full parse tree (also referred to as deep or constituency-based parse tree), obtained from constituency parsing (also called deep or full parsing). Another type of parsing exists, which is called shallow parsing (also called partial parsing or

[1] https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

chunking), where a less complex shallow parse tree is obtained. In shallow parsing, there are no embedded phrases i.e., no overlapping or recursive phrases.

4. *Dependency analyzer:* This module connects each word to the words that depend on it by creating a *dependency graph*, which is very useful for RE.

   In a dependency graph, the nodes are the words and the directed edges connect a word to the words that depend on it. An example of a dependency graph for the sentence: "Haifa, located 53 miles from Tel Aviv will host ICML in 2010", [3], is shown in Figure 2.3.

```
(ROOT
  (S
    (NP
      (NP (NNP Haifa))
      (VP (VBN located)
        (PP
          (NP (CD 53) (NNS miles))
          (IN from)
          (NP (NNP Tel) (NNP Aviv)))))
    (VP (MD will)
      (VP (VB host)
        (NP
          (NP (NNP ICML))
          (PP (IN in)
            (NP (CD 2010)))))))))
```

**Figure 2.2:** Parse tree of the sentence "Haifa, located 53 miles from Tel Aviv will host ICML in 2010"


**Figure 2.3:** Dependency graph of the sentence "Haifa, located 53 miles from Tel Aviv will host ICML in 2010"

## 2.3 Text Pre-processing

Text pre-processing has the goal of preparing a raw natural language text to facilitate its use in further IE tasks, e.g., NER. Text pre-processing involves steps of *cleaning*, *normalizing* and *annotating* the input text with linguist information. Segmentation in sentences and words (tokenization) is considered by several authors e.g., [12], as a text pre-processing step, although [10] considers it as an IE task, by itself.

The *cleaning* step consists on eliminating less useful parts of the text: punctuation (discussed in Section 2.1), stop words and noise, or special characters. Stop words correspond to very common words in a language e.g., "the" and "a", and typically correspond to function words that are not very useful. Text may contain noisy pieces of text or special characters that need to be removed. Lowercasing can be considered as a cleaning or normalization process that reduces data sparseness. If not performed,

capitalized words at the beginning of a sentence will be considered different from the same words, not capitalized, in the middle of the sentence. However, by making all words lowercase, some important information useful, for example, in NER regarding if a word is a proper noun or a common noun is lost.

*Normalization* is a typical text pre-processing step that processes and transforms tokens written differently into a single standard form. Other functionalities of text normalization include lemmatization and stemming. Both have the goal of representing words, that express the same meaning, by the same token. With stemming, words are transformed into their root (or stem), which is the most basic form of a word (without suffixes or prefixes). With lemmatization, words are transformed into their dictionary form.

*Annotation* [13] consists on the association and enrichment of the text with linguistic metadata, which is information that describes the text. Different NLP pre-processing libraries can be used to obtain various types of *linguistic annotations*: POS tags, parse tree and dependency graph. These *linguistic annotations* will be used as features during extraction, as already mentioned.

If the input text contains annotated entities, it is necessary to use an encoding scheme to assign a label to each token. The most widely used encoding scheme, and considered by some as standard, is the IOB (Inside, Outside and Beginning) encoding [14]: the "B" prefix is assigned to a token if it corresponds to the beginning of an entity, "I" if the token is a continuation of an entity and "O" for other tokens. In IOB-1 (typically just referred to as IOB), the "B" prefix is only used if two named-entities of the same type occur immediately after each other, where the first token of the second named-entity is assigned the "B" prefix. IOB-2, also referred to as BIO, is a derivative encoding scheme from IOB-1, where the "B" prefix is used for every token that is the beginning of a named-entity. Taking, as an example, the sentence, adapted from the WikiNER dataset (see Section 4.2.1):

*Tandis que le [comte de Toulouse]$_{Person}$ [Raymond VI]$_{Person}$ reçoit l'absolution, le [comte de Carcassonne]$_{Person}$ affronte seul l'armée.*

and that basically translates to: *While the count of Toulouse Raymond VI receives absolution, the count of Carcassonne confronts the army alone.* An IOB-1 and IOB-2 encoding scheme for the previous sentence and named-entity type *Person* is represented in Table 2.1. The first column corresponds to the token, the second column to the IOB-1 encoding and the third column to the IOB-2 encoding scheme. Moreover, "PER" corresponds to *Person* named-entity type.

## 2.4 Information Extraction Techniques

The classification of extracted information, i.e., named-entities (in the NER task) and relationships (in the RE task), can be performed through *rule-based methods* or *machine learning methods*. Classification (a type of pattern recognition [9]) consists on classifying an object into a certain class from a set of possible

| Token | IOB-1 | IOB-2 |
|---|---|---|
| Tandis | O | O |
| que | O | O |
| le | O | O |
| comte | I-PER | B-PER |
| de | I-PER | I-PER |
| Toulouse | I-PER | I-PER |
| Raymond | B-PER | B-PER |
| VI | I-PER | I-PER |
| reçoit | O | O |
| l’ | O | O |
| absolution | O | O |
| , | O | O |
| le | O | O |
| comte | I-PER | B-PER |
| de | I-PER | I-PER |
| Carcassonne | I-PER | I-PER |
| affronte | O | O |
| seul | O | O |
| l’ | O | O |
| armée | O | O |
| . | O | O |

**Table 2.1:** Example of IOB-1 and IOB-2 scheme

classes. Each object is characterized according to values of a selected set of features. A feature vector is built for each object. Furthermore, classification is done by identifying regularities, i.e., patterns, in the objects features. A system that automatically assigns a class to an object is called a *classifier*.

Classification can be binary or multi-class. In binary classification, an object is assigned to one of two classes. In multi-class classification there are more than two available classes. Multi-class classification can be further divided in: $(i)$ multi-label classification and $(ii)$ multinomial classification. In multi-label classification, an object can have more than one class assigned. The most common is multinomial classification, sometimes just referred to as multi-class classification, where classes are mutually exclusive, so each object is assigned to a single class from a set of possible classes. There are classification algorithms that perform multi-class classification, but most of the algorithms are binary. Different binary classifiers can be created for each class and combined to create multi-class classification in the so called *one-vs-all approach*. In IE, classification is considered multi-class and, most of the time, multinomial.

*Rule-based* IE methods perform IE through a collection of rules that typically are hand-coded by human experts or may be learned from examples. Experts need to have both domain and linguistic knowledge in order to be able to produce the extraction rules.

*Machine learning* IE methods can also be used. Machine learning algorithms have the ability to learn from data, by creating a mathematical model from it, and then make predictions based on this model. When labeled train data (generally, manually labeled) is available, supervised machine learning techniques can be used, however, when it is not, one can resort to unsupervised techniques. Futhermore, if the amount of labeled train data is limited, semi-supervised techniques can be used.

*Supervised machine learning*, and more specifically statistical classification, takes a dataset of input examples and tries to learn how to map a new example to the correct output class. The examples usually consist in feature vectors and their corresponding classes, that are obtained from annotated text, and form the train data or train set. In this context, a classifier has the goal of, given new input examples, called the test data or test set, predict their true class. To perform this, an appropriate algorithm needs to be chosen e.g., Support Vector Machine [15], as well as the features. There are two types of classifiers: generative and discriminative. Whereas generative classifiers try to model how a class can generate some input data (i.e., models the distribution of each class), discriminative classifiers try to learn what features discriminate between the classes (i.e., models the decision boundary between classes).

Supervised machine learning can be generalized to *structured prediction*. In *structured prediction*, the output is a complex structure like a sequence, graph, image, etc, unlike classification, where the output is a class. Sequence labeling or classification is a *structured prediction* problem where the input data is a sequence of objects and the output is a class sequence. In IE the sequence of objects typically correspond to a sequence of words.

Labeled data is generally labeled manually by humans. This is typically an expensive process, thus is not always available. *Unsupervised machine learning* uses unlabeled data to infer patterns on the data and learn about it. Finally, *semi-supervised machine learning* uses a combination of, typically, a small amount of labeled data and a large amount of unlabeled data. It is an alternative for situations where the amount of labeled data is limited, and a mix of *supervised machine learning* and *unsupervised machine learning* techniques can be used.

Specific techniques for NER and RE will be detailed in Section 3.1.2 and 3.2.2, respectively.

# 3

# Related Work

## Contents

In this Chapter, the most relevant related work for Named-Entity Recognition (NER) and Relationship Extraction (RE) is presented. Moreover, existing techniques and software tools are presented, for NER in Section 3.1 and for RE in 3.2.

## 3.1 Named-Entity Recognition

Named-entities [16, 17] are words or phrases that can be denoted with a proper name, and, that can be seen as part of a group of items with the same characteristics. They typically enclose names of people, names of organizations and names of locations. These named-entity types can be divided in finely-grained subtypes e.g., *Location* may be divided in *Country*, *State*, *City*, etc. Named-entities can be extended to include other domain-specific terms e.g., proteins and genes in the biomedical domain.

Named-Entity Recognition (NER) is a task that consists in the identification of named-entities in a natural language text and subsequent classification according to a pre-defined set of types of named-entities. Additionally, the NER community [16] generally agrees that temporal expressions and numerical expressions may also be extracted when performing this task. Moreover, the named-entity definition can be loosen to include other classes according to the tasks' needs e.g., book titles, phone numbers, animals, etc.

### 3.1.1 Features

Features of a word are clues that can be used for its prediction and classification. Features [11,12,16,17] that are typically used when performing NER can be divided in $(i)$ orthographic; $(ii)$ morphological; $(iii)$ lexical; and $(iv)$ contextual.

*Orthographic features*

Orthographic features are related to how the word is written. Features that are included in this category may be related to $(i)$ the word case e.g., if it starts with a capital letter, if it is all uppercase or if it is mixed case; $(ii)$ punctuation e.g., if it ends or has an internal period, if it has an internal apostrophe or hyphen; $(iii)$ digits e.g., if the word is an ordinal or cardinal number, if it contains digits, it it is a roman number; and $(iv)$ special characters e.g., greek letters, possessive mark.

*Morphological features*

Morphological features refer to the structure of the word and the way it can be decomposed in meaningful units i.e., morphemes. Named-entities often have common structures and units, so, features of this type are good indicators for recognizing and classifying named-entities. Examples of these features are the word's prefix and suffix (or a certain length), its singular version and its stem and lemma. Another

feature to be considered is the common word ending of certain named-entity types e.g., technological organization names often end in -soft or -tech. A way to consider common sequences of characters in the middle of words is to use *character n-grams* [18] as features. During training, a list of character n-grams and corresponding frequencies (or probabilities) for each named-entity type are computed. When predicting if a word is a named-entity and of what type, two kinds of features are generated for each existing named-entity type, based on the word's character n-grams: the probability of the word being a named-entity of the respective type, obtained by taking the average of the probabilities of each character n-gram of the word belonging to that type, and, the character n-gram of the word with the highest probability of existing in that named-entity type.

The word shape is also a useful feature that abstractly reflects the types of characters and their organization in the word, thus generating a pattern. An example of generating this pattern is to map all lower case letters to "x", all uppercase letters to "X", numbers to "d" and keeping the punctuation as is. Shorter word shape is another feature often used, where the idea described for the word shape is followed, but consecutive character types are removed. Considering the word "Paris" as an example, its word shape would be "Xxxxx" and its short word shape would be "Xx".

The part-of-speech (POS) tag of the word is also an important feature, because named-entities usually correspond to proper nouns. Features derived from the sentence's parse tree are also helpful (*syntactic features*) e.g., knowing if a word is part of a noun phrase is a good indicator of it being part of a named-entity.

*Lexical features*

The lexical usage of terms that are associated to named-entity types provide external knowledge to the NER system. These terms are contained within *lists*, and are also referred to as *gazetteers*, *lexicons* or *dictionaries* [16]. As an example, we can think of a *list* of locations: if a word belongs to this list, than the probability of it being a *Location* named-entity is high.

The term *gazetteer* is often used when referencing specifically to lists of locations.

Lists of names i.e., first names and surnames, and organizations are also used, but may not always be useful [11].

Most approaches require that words match exactly one element of the given *list*. However, some flexibility may be allowed by considering stemmed versions of the words, or by the words being "fuzzy-matched", where a similarity measure between the word and the list's words is applied and a threshold is defined, etc.

Lists of predictive words applied to the surrounding words of the given word, may also be an indicator of the named-entity type. Examples of this are preceding and following titles e.g., "Mr.", or other terms like "Inc.", which are good indicatives of *Person* and *Organization*.

*Contextual features*

The context of a named-entity is also a good indicator that the word is a named-entity, thus it can be used as a feature. The context of a word is obtained by defining a context window of a certain size, that will capture the preceding and succeeding words, i.e., neighboring words, of each word. Not only the words are captured, but also some of their features, allowing to induce certain regularities for when a named-entity occurs. Moreover, each feature of a neighboring word is a feature of the word that is being predicted. Another approach is to create features that are the *conjunction* of features of the neighboring words.

### 3.1.2 Techniques

NER can be seen as a sequence labeling or sequence classification task, where given as input a sequence of words, e.g. a sentence, a sequence of labels or classes, i.e., named-entity types, is returned as output. The labels will capture not only the type of the named-entities, but also their boundary. Formally, the sequence of words will be denoted by $X = \{x_1, ..., x_n\}$, where $n$ is the number of words, and the sequence of corresponding labels by $Y = \{y_1, ..., y_n\}$ where each label $y_i$ belongs to the pre-defined set of named-entity types $\mathcal{Y}$ which includes a special type for when a token is not a named-entity.

To label the tokens, an encoding scheme that captures the fact that named-entities may be formed by multiple tokens is necessary. The encoding scheme should encode the boundaries and the labels of each named-entity in the original text. Approaches typically use the IOB or BIO notation that was explained in Chapter 2.

The different techniques that exist for NER can be grouped in the following standard methods [11]:

1. **Supervised methods** that require labeled training data and can be further divided into:

   (a) **Feature-based methods** where features are designed from each word that is then represented by a feature vector. These are used to train either a multi-class classifier or a sequence model/classifier, that will later be used to make predictions on new data.

   (b) **Neural-based methods** where deep learning neural models are capable of automatically learning features when given raw data thus are not dependent on hand-crafted features.

2. **Rule-based methods** that use hand-crafted rules based on lexical features and domain knowledge to perform NER.

In the following Sections we will give more details regarding the supervised techniques, that use labeled data, do not require linguistic knowledge, and that better fit the scope of this thesis.

#### 3.1.2.1 Feature-based Methods

In feature-based techniques, each word is usually be described by a feature vector, that contains several attributes of the word i.e., features, like the ones presented in Section 3.1.1. Moreover, these features can be specified by boolean, numerical or nominal values.

One solution for the NER sequence labeling problem would be to independently classify each token of the input sequence using a conventional classifier. However it is a sub-optimal solution, because context would not be taken into account. In other words, the optimal label of a certain token of the sequence should depend on the labels of the neighboring tokens, because labels have conditional dependencies. Taking the example in [19], we can look at the sequence "Paris Hilton", where "Paris" can be either a *Location* or a *Person* and "Hilton" can be either an *Organization* or a *Person*. If one of them is labeled as *Person*, the other should be labeled as *Person* as well. Thus, as we can see, the labeling decisions influence each other. That is where sequence models, also called sequence classifiers, are useful, because when they are classifying a token, they take into consideration the labels previously assigned to the preceding tokens.

Different algorithms have been used over the years for performing NER. Considerable work was done using Hidden Markov Models (HMM), Decision Trees, Maximum Entropy Models (ME), Support Vector Machines (SVM) and Conditional Random Fields (CRF). Decision Trees and SVM are conventional classifiers and do not take into account the context of the words when classifying, so, the rest of this section will be focused on sequence models i.e., HMM, MEMM and CRF.

*HMM*

Hidden Markov Model (HMM) [20–22] is a sequence model that extends Markov Chains, which are sequential stochastic models which means they are defined by a sequence of random variables, states, $Q = \{q_t\}$, whose outcomes are not certain. In a Markov Chain, states correspond to observable events and it is possible to predict the probability of a sequence of states e.g., probability of a sequence of words. But, in some cases, it is necessary to predict the probability of events that are not observable, i.e., that are *hidden*. The concept of a Markov Chain can be extended to have has two stochastic processes, defining the model know as HMM: one defined by the sequence of hidden events, i.e., hidden states, and another of observable events through which the hidden events are observed.

An HMM is specified by several components:

- Sequence of $T$ states, $\boldsymbol{Q = \{q_t\}}$, whose values belong to the finite set of $N$ values $S = \{S_1, ..., S_N\}$, known as *state space*
- Sequence of $T$ observations, $\boldsymbol{O = \{O_t\}}$, whose values belong to the finite set of $M$ values $V = \{v_1, ..., v_M\}$, known as *observation space*. The observations represent the physical output of the

model

- Transition Probability Matrix $\boldsymbol{A}$, represents the dynamics of the Markov Chain. Its size is of $N \times N$. $[A]_{ij}$ is the value in row $i$ and column $j$, which represents the probability $a_{ij}$ of transitioning from state $i$ to state $j$,

$$a_{ij} = P(q_t = S_j | q_{t-1} = S_i)$$

- Observation Probability Matrix $\boldsymbol{B}$, of size $N \times M$. $[B]_{jk}$ is the value in row $j$ and column $k$, which represents the probability $b_j(v_k)$ of observing observation $k$ on state $j$,

$$b_j(v_k) = P(O_t = v_k | q_t = S_j)$$

These probabilities are also called *emission probabilities*.

- Initial State Distribution $\boldsymbol{\pi}$, represents the probability of the process starting in each state.

$$\pi_i = P(q_1 = S_i)$$

The model makes two strong assumptions:

1. *Markov Assumption*: the probability of a state only depends on the previous state, and not on all of its preceding states:

$$P(q_t = S_j | q_1 = S_d, ..., q_{t-1} = S_i) = P(q_t = S_j | q_{t-1} = S_i) \tag{3.1}$$

2. *Output Independence*: the probability of an observation only depends on the state that produced it, in other words, to predict an observation at time step $t$, only the state at time step $t$ is necessary, and not all of its preceding states or observations:

$$P(O_t = v_k | q_1 = S_d, ..., q_t = S_j, O_1 = v_x, ..., O_{t-1} = v_z) = P(O_t = v_k | q_t = S_j) \tag{3.2}$$

An HMM can be described by the its model parameters using the notation $\lambda = (A, B, \pi)$. Since training data is available, the probabilities of the parameters can be estimated from the data.

For NER, we want to solve the problem of given a sequence of words (observations), discover the most probable sequence of labels (states), using an HMM (called the *decoding task*). This problem can be solved using a dynamic programming method called the *Viterbi algorithm* [23], which is the most common decoding algorithm for HMMs.

The joint probability of a state sequence $Q = \{q_1, ..., q_T\}$ and an observation sequence $O = \{O_1, ..., O_T\}$

is the probability of $Q$ and $O$ occurring simultaneously, given the model $\lambda$, and can be computed as follows:

$$P(Q, O|\lambda) = P(O|Q, \lambda) \cdot P(Q, \lambda)$$

In order to discover the best or most probable state sequence $Q = \{q_1, ..., q_T\}$ for a given observation sequence $O = \{O_1, ..., O_T\}$, the algorithm computes the joint probability of $Q$ and $O$, given the model $\lambda$:

$$\delta_t(i) = \max_{q_1,...,q_{t-1}} P(q_1, ..., q_t = S_i, O_1, ..., O_t|\lambda)$$

which is the probability of the path, i.e., state sequence, with highest probability at time step $t$, thus accounting for the first $t$ observations and ending in state $q_t = S_i$. For a state $q_t = S_j$ the value of $\delta_t(j)$ is, by induction:

$$\delta_t(j) = \max_i[\delta_{t-1}(i) \cdot a_{ij}] \cdot b_j(O_t)$$

where three factors are multiplied: the previous Viterbi path probability $\delta_{t-1}(i)$, obtained from the previous time step, the transition probability from the previous state $S_i$ to the current state $S_j$, $a_{ij}$, and the probability of observing the current observation $O_t$ given the current state $S_j$, $b_j(O_t)$. $\delta_t(j)$ represents the probability of the model being in state $S_j$ after observing the first $t$ observations and going through the most probable state sequence up to $q_{t-1}$. For each state $S_j$ possible at each time step $t$, $\delta_t(j)$ is computed recursively by taking the most probable state path that could lead to state $S_j$.

To obtain the state sequence, it is necessary to compute, along with $\delta_t(j)$, the value $\psi_t(j)$ which corresponds to the argument $i$ that maximized $\delta_t(j)$ i.e., the state $S_i$ that led to the state $S_j$. Moreover, the best state sequence i.e., best sequence of labels, for the observation sequence i.e., word sequence, is computed by backtracking the best state path to the beginning using $\psi$, thus also assigning maximum likelihood to the sequence of observations.

A representation of an HMM computing the probability of the correct sequence of states (labels) for the sentence "Mark Watney visited Mars" is shown in Figure 3.1. The arrows in Figure 3.1 reflect the dependencies between the variables and are associated with a probability. As we can see, the probability of an observation depends only on the current state, the state that produced it. Additionally, the probability of the current state depends only on the previous state.

*MEMM*

Maximum Entropy Markov Models (MEMM) [24, 25] are an extension of Maximum Entropy (MaxEnt or ME) models. ME models, also known as multinomial logistic regression, independently assign a label

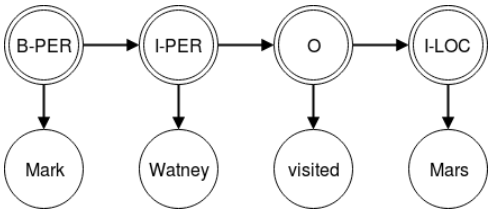


**Figure 3.1:** HMM representation of label sequence computation for a sentence

to an observation by determining a probability. MEMMs extend the ME model to sequence labeling, based on HMMs.

An HMM is a generative model which makes it hard to directly incorporate features. The model rests on two probabilities: the state transition probability, $P(q_t|q_{t-1})$, and the observation probability, $P(O_t|q_t)$. Thus, to incorporate features in the model, they would need to be encoded, in some way, in one of the two probabilities.

The goal is to find the most probable state sequence $Q$ given an observation sequence $O$, $P(Q|O)$. With an HMM, instead, we compute the probability of a state producing a certain observation, $P(O|Q)P(Q)$. Since the observations are given, the probability of the observations is not useful; we are solely interested in estimating the probability of the state sequence generated by the observations.

MEMMs are an alternative to HMM that can compute the $P(Q|O)$ directly. They are discriminative models and are able to directly discriminate amid possible state sequences. A MEMM models only one probability: state-observation transition probability $P(q_t|q_{t-1}, O_t)$, the probability of the current state $q_t$ given the previous state $q_{t-1}$ and the current observation $O_t$. Observations that are associated to states when using HMMs, are associated to state transitions when using MEMMs.

Moreover, HMMs employ is a Transition Probability Matrix $A$ and an Observation Probability Matrix $B$. In contrast, when using MEMMs, we have a single matrix, of size $(N \times M) \times N$, with the probabilities of all possible combinations of previous states $q_{t-1}$ and current observations $O_t$ with current states $q_t$.

The use of state-observation transition probability in MEMMs (instead of state transition and observation probability in HMMs) enables the modeling of transitions in terms of multiple, non independent features of observations. Maximum Entropy is used to estimate the conditional probability of a state i.e., label, given its previous state, the current observation i.e., word, and any desired features, exponentially, as follows:

$$P(q_t|q_{t-1}, O_t) = \frac{1}{Z(O_t, q_{t-1})} \exp\left(\sum_{i=1}^{N} \lambda_i f_i(O_t, q_t)\right)$$

where $Z(O_t, q_{t-1})$ is the normalization factor (which makes the distribution sum to $1$ across all next states $q_t$), $\lambda_i$ are feature weights to be learned from the training data and $f_i$ is the probability of feature $i$ given the current observation $O_t$ and a possible new current state $q_t$.

To perform the decoding task i.e., discover the most probable state sequence given an observation sequence, the solution is to use the Viterbi algorithm like in HMMs, however an adaptation needs to be performed. In order to discover the best or most probable state sequence $Q = \{q_1, ..., q_T\}$ for a given observation sequence $O = \{O_1, ..., O_T\}$, using an HMM, the algorithm computes:

$$\delta_t(j) = \max_i [\delta_{t-1}(i) P(q_t = S_j | q_{t-1} = S_i)] \cdot P(O_t = v_k | q_t = S_j) = \max_i [\delta_{t-1}(i) a_{ij}] \cdot b_j(O_t)$$

With a MEMM, the transition probability, $a_{ij}$ or $P(q_t = S_j | q_{t-1} = S_i)$, and the observation probability, $b_j(O_t)$ or $P(O_t = v_k | q_t = S_j)$, are replaced by the probability $P(q_t | q_{t-1}, O_t)$:

$$\delta_t(j) = \max_i \delta_{t-1}(i) P(q_t = S_j | q_{t-1} = S_i, O_t = v_k)$$

Training in MEMMs is based on the same algorithm used for multinomial logistic regression, where the weights or parameters are trained to maximize the log-likelihood of the training data.

A representation of an MEMM computing the probability of the correct sequence of states (labels) for the sentence "Mark Watney visited Mars" is shown in Figure 3.2. The arrows in Figure 3.2 reflect the dependencies between the variables and are associated with a probability. As we can see, the probability of a state depends on the previous state and the current observation.

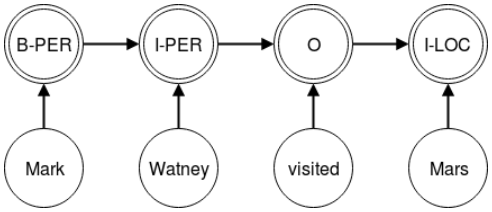


**Figure 3.2:** MEMM representation of label sequence computation for a sentence

*CRF*

Markov models, including MEMMs, have a weakness called the *label bias problem* i.e., outgoing transitions of a state only compete against each other, because the state transitions are normalized locally, as opposed to against all other transitions in the model. If a state only has one outgoing transition, it is obliged to move to next state, making the observation irrelevant. Consequentially, states with fewer outgoing transitions are preferred.

Conditional Random Field (CRF) [26–28] is a sequence model that improves over MEMM, solving the *label bias problem*. MEMM uses an exponential model for each state to describe the next states, meaning there is a per-state normalization. CRF uses a single exponential model for the joint probability of the entire state sequence given the observation sequence, which enables the state transition

probabilities to be globally normalized.

Consider that $X$ and $Y$ are random variables that, respectively, represent observation sequences (i.e., word sequences) and corresponding state sequences (i.e., label sequences). CRFs can be seen as a discriminative undirected graphical model, globally conditioned on $X$. Consider $G = (V, E)$ is an undirected graph with a node $v \in V$ for each element $Y_v$ of $Y$. $(X, Y)$ is a CRF if it satisfies the *Markov property* with respect to the graph, when conditioned on $X$:

$$P(Y_v|X, Y_w, w \neq v) = P(Y_v|X, Y_w, w \sim v)$$

where $w \sim v$ means that $w$ and $v$ are neighbors in the graph $G$. Linear-chain CRFs correspond to a graph structure where the nodes, that correspond to the elements of $Y$, form a simple first-order chain. Linear-chain CRFs are the simplest and most commonly used graph structure when modeling sequences, to which we will refer to simply as CRF for the rest of the document.

A CRF is a model defined by the probability distribution of a state sequence $y$ given an observation sequence $x$ that takes the form:

$$P(y|x, \lambda) = \frac{1}{Z(x)} \exp\Big(\lambda \cdot F(y, x)\Big)$$

where $Z(x)$ is a normalization factor, $\lambda$ are feature weights to be learned from the training data and $F$ is the "global" feature vector, because it maps the entire state sequence to a *d*-dimensional feature vector. The global feature vector $F$ can be decomposed in:

$$F(y, x) = \sum_{n=1}^{N} f(y_{n-1}, y_n, x, n)$$

meaning the $F$ global feature vector is obtained by summing $f$ over the $N$ different state transitions in $y_1, ..., y_N$. The feature vector $f$ analyzes the entire $x$ sequence, the current state $y_n$, the previous state $y_{n-1}$ and the current position $n$ in the sequence.

We call $F_k$ the $k$'th "global" feature vector for $k = 1, ..., d$ and is obtained by

$$F_k(y, x) = \sum_{n=1}^{N} f_k(y_{n-1}, y_n, x, n)$$

The decoding problem in a CRF i.e., finding the most probable (or best) state sequence i.e., label sequence, given an observation sequence i.e., word sequence, is defined by the equation:

$$\underset{y}{\operatorname{argmax}}\, P(y|x, \lambda)$$

which can be simplified into:

$$\underset{y}{\operatorname{argmax}} \sum_{n=1}^{N} \lambda \cdot f(y_{n-1}, y_n, x, n)$$

The problem can be solved using a variant of the Viterbi Algorithm, where $\delta_t(j)$ becomes:

$$\delta_t(j) = \max_i \delta_{t-1}(i) + \lambda \cdot f(i, j, x, t)$$

During the CRF training phase, the weights or parameters that best fit the training data are discovered, i.e., the weights that maximize the log-likelihood of the training data. The standard approach uses iterative scaling or gradient-based methods [29].

A representation of an CRF computing the probability of the correct sequence of states (labels) for the sentence "Mark Watney visited Mars" is shown in Figure 3.3. CRF is represented by an undirected graph and shows that the probability distribution of a sequence of states (labels) is given by a sequence of observations.

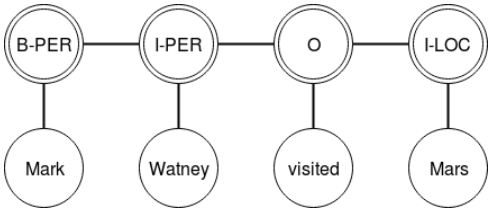


**Figure 3.3:** CRF representation of label sequence computation for a sentence

#### 3.1.2.2 Neural-based Methods

Deep learning [19, 30] is a family of machine learning methods, that are composed of many processing layers, typically artificial neural networks. Artificial neural networks attempt to simulate the human brain and contain a series of interconnected artificial neurons (or units) arranged in layers, which have associated *weights*. The *input layer* receives information of a certain format, processes and learns about it in *hidden layer*, and, an *output layer* that makes a decision or prediction about the input. A deep artificial neural network is composed of more than one hidden layer.

Deep learning methods use deep artificial neural networks, and are not only able to learn to make predictions but also able to transform the input data to its most suitable representation for prediction i.e., learn features directly from the data. This is achieved by feeding the data into the network that will then successively transform it until the output is predicted in a final transformation. Then, the errors are propagated back through the network, adjusting the network's *weights*.

Deep learning methods are independent from hand-crafted features, that require some engineering, and from lexical features that are domain specific, thus making them domain independent. While the

previous feature-based techniques presented generate *linear* mappings, neural-based methods can benefit from *non-linear* mappings, that allow the model to learn more complex features.

Recent methods for NER are deep learning based or neural based, and achieve state-of-the-art results. The methods combine multiple neural networks of different architectures, but all tend to use a Recurrent Neural Network (RNN) which is a kind of neural network architecture specialized for sequential data. This neural network is trained to produce informative representations of the data to be fed to another neural network or another network component, that will predict the labels.

Furthermore, we can say that the general architecture of a deep-learning based NER model is composed of an *embedding layer* (input layer), a *context encoding layer* and a *sequence labeling layer* (output layer). These will be further explained below.

*Embedding layer*

When using deep learning to solve NLP problems, typically, an *embedding layer* is used, also known as *lookup layer*. An *embedding layer* maps the input sequence of words to a sequence of vectors which are distributed representations of the words i.e., *word embeddings* [31].

The input given to the neural network will be words, that can be seen as categorical or symbolic features (words from a defined vocabulary $V$). Moreover, the embedding layer of the neural network will map the categorical features to *d*-dimensional vectors, for some $d$, that are considered parameters of the model (and that can be trained in conjunction with them). These vectors are distributed representations of each word. They represent words in low dimensional real-valued dense vectors. Each dimension of the vector is a latent feature that captures syntactic and semantic properties of the word.

The mapping consists in a lookup operation, that takes a word and accesses a matrix with it to obtain its *word embedding*. The matrix, also called embedding matrix, contains a collection of $|V|$ vectors, one for each word in the vocabulary, and is of the size $|V| \times d$.

*Context encoding layer*

A *context encoding layer* captures the context dependencies from the input representations and produces context-dependent representations. Usually a RNN is used to achieve this, which is capable of analyzing sequential data. A simple RNN has an input layer $x$, an hidden layer $h$ and an output layer $y$ as it is possible to see in Figure 3.4. Furthermore, it earns the name "recurrent" because it makes the same computation for every element in the input sequence which is dependent on the previous computations, depicted by the loop in Figure 3.4.

In Figure 3.4 we can also see the RNN's recursion being unrolled, which basically means we can look at the network for the complete sequence i.e., if the sequence contains $n$ words, then the unrolled network will contain $n$ layers, one for each word. The RNN takes as input an ordered sequence of $n$

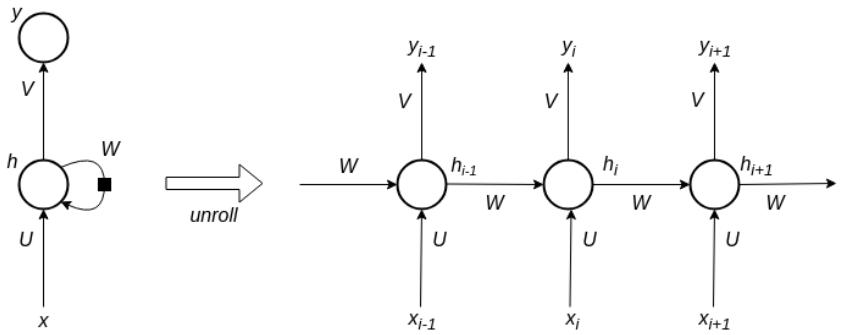


**Figure 3.4:** Graphical representation of a typical RNN and an RNN being unrolled

*d*-dimensional vectors for every word, which usually are *word embeddings*. Moreover, the input at time step $i$ is $x_i$. The hidden layer returns a vector $h_i$ with context information about the sequence for every step $i$ in the input, by means of weight parameters, $U$ and $W$, the input vector $x_i$ and the previous hidden vector $h_{i-1}$. And, the output layer returns a vector $y_i$, also for every step $i$ which contains the probabilities across the set of labels (and makes use of the weight parameter $V$).

A RNN is defined recursively, by means of a non-linear activation function $f$, typically a *sigmoid* or *tanh* function, that takes as input the hidden vector $h_{i-1}$ and an input vector $x_i$ (and associated weight parameters, $U$ and $W$) and returns a new hidden vector $h_i$:

$$h_i = f(Ux_i + Wh_{i-1})$$

where $U$ and $W$ are the recurrent layer weight parameters that are computed during training.

The network outputs a matrix of scores $P$. This matrix is composed of the network's output vectors, obtained from the output layer $y$, which represents the probability distribution over labels for each step $i$ in the input.

$$y_i = softmax(V \cdot h_i),$$

where $V$ is a weight parameter computed during training. The *softmax* function computes the probability distribution over the set of labels. $P_{ij}$ is the score of the $j^{th}$ label of the $i^{th}$ word in the sequence.

Thus, there exists a connection between the previous hidden vector and the current hidden vector as well the weight parameters. We can say that this hidden or recurrent layer stores history information which in theory allows the prediction of the output based on long range dependencies in the data. However, in practice, RNNs fail to learn long range dependencies and are biased towards the most recent sequence inputs.

Long Short-Term Memory (LSTM) networks are a RNN variant that incorporate a memory-cell and, thus, are capable of capturing long range dependencies. It is achieved by the use of multiple gates that

control the proportion of the input that should be given to the memory cell and the proportion of the previous hidden vector that should be forgotten.

With the RNN or the LSTM, an hidden vector $h_i$ of a word $i$ captures its left context. With a bidirectional LSTM (bi-LSTM) network, both past features and future features can be taken into account for prediction. A bidirectional LSTM consists on using two LSTMs, one that operates as described before and that reads the input sequence in a left-to-right manner (*forward* LSTM), and another that reads the sequence in reverse order, that captures the right context of a word (*backward* LSTM). A word's hidden vector $h_i$ is obtained by concatenation of each hidden vector generated by each LSTM, i.e., the left context vector $h_i^{left}$ and the right context vector $h_i^{right}$.

*Sequence labeling layer*

The *sequence labeling layer* predicts the labels of the words in the original sequence. Each final output $y_i$ label is predicted using the score matrix $P$ output by the *context encoding layer*, or more specifically, the bi-LSTM network, in the so called output layer or *sequence labeling layer*. The output labels $y_i$ can be computed directly from the score matrix, but local choices would be made.

CRFs consider sentence level label information, and thus maximize the label probability for the complete sentence. They typically have higher accuracy in labeling tasks, thus are widely used in this context. The use of this layer will also allow using past and future labels to predict the current label. This achieved by using a state transition matrix $A$, where $A_{ij}$ corresponds to the score of transitioning from state (label) $i$ to the state $j$. Considering $y$ is a sequence of labels for the input sequence of words $x$, its score is given by:

$$score(y) = \sum_{i=0}^{n} A_{y_i, y_{i+1}} + \sum_{i=1}^{n} P_{i, y_i}$$

The Viterbi Algorithm can then be used to discover the label sequence $y$ that has the maximum score.

*General Architecture*

The standard neural model for NER is to combine a bidirectional LSTM network with a CRF to form a BI-LSTM-CRF model. The architecture of this model is presented in Figure 3.5, adapted from [32]. The model receives as input a sequence of words that are mapped to a sequence of word vector representations e.g., *word embeddings*, in the *embedding layer*. These are given to a bi-LSTM which generates two context vectors i.e., left context vector $l_i$ and right context vector $r_i$, that are concatenated in a vector $c_i$, in the *context encoding layer*. This context vector is given to the *sequence labeling layer*, i.e. CRF layer, and a label $y_i$ is returned for every word.

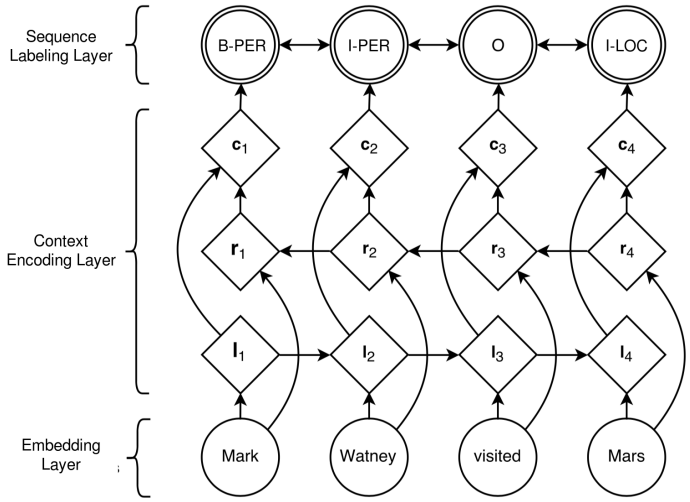


**Figure 3.5:** Architecture of a BI-LSTM-CRF model

One of the first works that uses a bi-LSTM-CRF architecture for sequence labeling tasks, including NER, is [33]. Their word embeddings are combined with hand-crafted features, including orthographic features and context features. These hand-crafted features are passed directly to the output layer, the CRF, instead of passing through the bi-LSTM layer. This accelerates the training while having similar accuracy. A similar model was applied by [32], where no hand-crafted features were used. They use character embeddings learned during training and concatenate them with word embeddings initialized with externally pre-trained embeddings to obtain their actual word embedding.

More recent state-of-the-art works use deep learning for NER with new word embedding techniques. Traditional word embeddings (e.g., Word2Vec [31], Glove [34] and fastText [35]) are static, meaning that a word's representation is the same no matter its context i.e., surrounding words. New word embedding techniques are dynamic, in the sense that the word's representation is dependent on its context i.e. the same word in different contexts will have different representations. BERT (Bidirectional Encoder Representations from Transformers) [36] is a new distributed representation approach that achieves state-of-the-art results for NER. It pre-trains bidirectional representations of words' contexts i.e., conditions on both left and right context. Another state-of-the-art approach for NER uses [37] *flair embeddings* that are character-level word embeddings dependent on context.

### 3.1.3 Tools

There are several tools available that are capable of performing NER. These tools can be black-box web services or third-party libraries that provide pre-trained models or that enable to train a model. In the rest of this section we present the NER tools we consider most relevant.

Other tools we found include NLTK[1](Natural Language Toolkit) is a Python library for NLP. Among other tasks, NLTK is capable of performing NER. NLTK supports Stanford NER, however it has its own NER model. This model uses a Maximum Entropy (ME) algorithm and uses several features that help during the extraction. It has no support for the French language. We also found more black-box web services that we will be not included in this section.

Moreover, we also found Polyglot[2] [38] that an NLP library capable of performing, among other tasks, NER, with minimal human intervention and knowledge. It creates models using data generated by distant supervision using Wikipedia and Freebase for $40$ languages, including French.

#### 3.1.3.1 Stanford NER

Stanford NER is a NER Java software tool that implements a CRF sequence model, along with a feature extractor and options to define new features. It is also available in other programming languages through modules, wrappers, interfaces or packages.

Additionally, it is included in Stanford CoreNLP [39], which is an annotation pipeline system that provides other essential NLP tools, like the Stanford Parser or the Stanford POS tagger.

The CRF implemented in the Stanford software is similar to the baseline local+Viterbi model in [40]. Here, the label decision at a particular position of the sequence only depends on a small local window, as usual, and Viterbi is used to compute the most likely label sequence.

Different features are extracted [3], including context features (e.g., words in a window), morphological features (e.g., character n-grams), label sequences and conjunctions of features. Lexical features, more specifically *gazetteers*, may also be used, and are included as files. Distributional similarity clustering can also be used, where words are clustered based on their similarity context distributions. Moreover, clusters contain (semantically or syntactically) similar words and the corresponding cluster IDs are used as features, allowing the NER system to generalize better. There are several other features and parameters that can be specified.

Furthermore, Stanford NER distribution includes several pre-trained models, but none for the French language. However, it enables to train a new model using labeled data, and specifying the features to be extracted or adding new features.

#### 3.1.3.2 SpaCy

SpaCy is a free, open-source Python library for NLP, capable of performing NER. The SpaCy NER system uses a word embedding strategy with subword features and blossom embeddings, a deep Con-

[1] https://www.nltk.org
[2] https://polyglot.readthedocs.io
[3] www.nlp.stanford.edu/software/jenny-ner-2007.pdf and
www.nlp.stanford.edu/nlp/javadoc/javanlp/edu/stanford/nlp/ie/NERFeatureFactory.html

volutional Neural Network (CNN) with residual connections and a transition-based named-entity parsing.

SpaCy approaches deep learning for NER in four steps[4]: embed, encode, attend and predict. In the *embed* step, words are represented by context-independent dense vectors, called blossom embeddings. Hashed embedded representations of subword features are concatenated and fed to a multi-layer perceptron to get a vector for each word. Afterwards, in the *encode* step, the word embeddings are encoded into context-sensitive representations using a trigram CNN. Residual connections are used, that make the output of each convolutional layer of the CNN to be the sum of the actual output and its input. The the goal of the *attend* step is to take the ouput of the previous step, a vector per word, and compute a single summarized vector. The final step is the *prediction* step that uses a standard multi-layer perceptron to predict the labels.

A transition-based system based [32] is used, where instead of labeling each word, the model directly constructs representations of multi-token named-entities.

SpaCy has a pre-trained model for French that is capable of performing several tasks including NER (trained with the WikiNER dataset [41]). It is capable of detecting *Person*, *Location*, *Organization* and *Miscellaneous* named-entities.

#### 3.1.3.3 Apache OpenNLP

Apache OpenNLP is a machine learning open-source Java library for NLP tasks, including NER, that is available through the Name Finder API. It offers several pre-trained models for English, Spanish and Dutch, but not for French.

It is also possible to train new models using OpenNLP by providing it with data in the OpenNLP's training format. OpenNLP uses, by default, a Maximum Entropy (ME) model which can be changed to a Perceptron or Naive Bayes. It is also possible to specify the number of iterations and the cutoff value (named-entities that are found less times that the defined cutoff are ignored). Certain features are extracted from the training data to be used by the model.

#### 3.1.3.4 NeuroNER

NeuroNER [42] is an open-source program available online, that performs NER. It requires Python 3.5, TensorFlow 1.0 and *scikit-learn*[5] and optionally BRAT, and can run either from the command line or from a Python interpreter.

NeuroNER is composed of two main components: a NER engine (NER engine) and an interface with BRAT. BRAT[6] is a web-based tool for text annotation, that enables creating, changing or viewing annotations. The NER engine receives as input three sets of labeled data: train, validation and test sets.

[4] https://youtu.be/sqDHBH9IjRU
[5] https://scikit-learn.org
[6] http://brat.nlplab.org

An LSTM is used in the NER engine, composed of three layers: the character-enhanced token-embedding layer, the label prediction layer and the label sequence optimization layer. The *character-enhanced token-embedding* layer maps each token in a sequence to character-enhanced token embeddings, by combining word and character embeddings. The sequence of character-enhanced token embeddings is passed to the *label prediction* layer that outputs the probability distribution over the set of labels for each token. Finally, based on the sequence of probability vectors, the *label sequence optimization* layer outputs the most probable sequence of labels for the original sequence of tokens. All layers learn jointly and the user can specify several parameters to be used during the training process.

NeuroNER supports monitoring of the network during the training process, via plots generated by NeuroNER itself, showing the learning curve of the model, and via TensorBoard (TensorFlow's visualization toolkit), showing the real time performance of the model over the test set.

NeuroNER also provides to the user a set of pre-trained models and pre-trained token embeddings. None of these are in French. It allows a user to train their own model by providing labeled data and, optionally, token embeddings.

#### 3.1.3.5 Flair

Flair [4] is an open-sourced NLP library developed by Zalando Research[7]. It achieves state-of-the-art performance in a range of tasks, including NER. It is implemented in Python and builds on top of PyTorch [43], considered one of the best deep learning frameworks.

Current approaches to sequence labeling combine different types of word embeddings, in particular, dynamic contextual word embeddings combined with traditional static word embeddings and show improved results [4,37]. The combination of word embeddings requires engineering effort and to solve this problem the Flair framework was proposed. Flair's most prominent feature is its interface that facilitates that allows combining or stacking different word embeddings, by concatenating them. It includes popular word embeddings like GloVe [34], BERT [36] and many others. Moreover, the Flair framework also features their Flair embeddings [37].

Moreover, and in the context of NER, when training a sequence labeling model, the word embeddings are passed to a bi-LSTM-CRF.

They make multiple pre-trained models available for several tasks including a French NER model, that outperform previous state-out-the-art results. Moreover, the French NER model was trained on the WikiNER dataset [41] using French character embeddings, trained on Wikipedia, combined with French fastText embeddings [35]. It is capable of detecting *Person*, *Location*, *Organization* and *Miscellaneous* named-entities.

[7] https://research.zalando.com/

#### 3.1.3.6 IBM Watson NLU

IBM Watson Natural Language Understanding (IBM Watson NLU) is a web service with several NLP capabilities that enable to analyze text and extract metadata, such as named-entities (and relationships between entities). Additionally, it supports the French language.

To use the NLU API, an HTTP Post request needs to be made to the service using an API key. The IBM Watson Knowledge Studio[8] can be used to extend the NLU with customized models that can identify customized information.

The free version of the service supports the extraction of $30,000$ NLU items per month, where one NLU item corresponds to a group of $10,000$ characters and a feature e.g., named-entities. It also limits the use of only one customized model. Other service plans can be acquired to increase the values or remove the constraints.

#### 3.1.3.7 Open Calais

Open Calais is a black-box web service developed by the Text Metadata Services (TMS) group at Thomson Reuters. Moreover, it attaches semantic metadata tags to text that it receives as input. It uses a combination of NLP, machine-learning and customized pattern-based methods for processing the input text and extracting semantic information. Among other processes, it performs NER and RE, based on a pre-defined set of named-entities and relationships.

It supports French and is able to extract named-entities in French texts, however, it is not able to do the same for relationships. Open Calais requires an API access token that is used to make an HTTP Post Request that sends the desired input document and obtains its semantic metadata tags.

It supports $5,000$ requests per day, each with a maximum request input size of $100KB$. Moreover, about a second should be left between requests. It is possible to increase these values by subscribing to other versions of Open Calais.

Open Calais detects each named-entity in the text but is also capable of grouping multiple named-entity instances that refer to the same named-entity, by comparing the instances text strings. For French input text, it is able to extract named-entity types such as: City, Country, Organization, Person. It is also capable of generating a confidence score for some types of named-entIties.

#### 3.1.3.8 Discussion

Table 3.1 summarizes the most relevant NER tools. We found multiple black-box web services capable of performing NER. We can state that most of the black-boxes support multiple languages. Some of those

[8] https://www.ibm.com/watson/services/knowledge-studio/

| | Black-box | Third-party library | French support |
|---|---|---|---|
| Stanford NER | | x | |
| SpaCy | | x | x |
| Apache OpenNLP | | x | |
| NeuroNER | | x | |
| Flair | | x | x |
| NLTK | | x | |
| IBM Watson NLU | x | | x |
| Open Calais | x | | x |

**Table 3.1:** NER tools

include IBM Watson NLU and Open Calais, that are capable of, among other languages, of dealing with French texts.

We found several third-party libraries: Stanford NER, SpaCy, Apache OpenNLP, NeuroNER and Flair. Only Flair and SpaCy have available pre-trained NER models for French.

Moreover, Stanford NER and Apache OpenNLP allow training models that implement a feature-based technique. These types of techniques require feature engineering. Additionally, NLTK supports training Stanford NER models.

SpaCy, NeuroNER and Flair allow training recent neural-based state-of-the-art techniques. NeuroNER uses a combination of character embeddings with pre-trained word embeddings (that can be specified before training, e.g., GloVe [34], fastText [35], etc) to train a LSTM with or without a CRF in the output layer. Flair allows combining various word embeddings that are passed to a bi-LSTM-CRF. The architecture used by NeuroNER can be achieved by using Flair, except for the fact that NeuroNER uses a LSTM instead of a bi-LSTM, whose bi-directionality has proven to be better for understanding context. Out of the third-party libraries presented, we consider that Flair and SpaCy have the potential to train better performing models when compared to the other libraries' techniques.

## 3.2 Relationship Extraction

Relationship Extraction (RE) is an IE task whose aim is to identify semantic relationships between two or more entities in natural language text. Most work on RE focuses on binary relationships, which are relationships that occur between two entities. Multi-way relationships, occur between three or more entities. Moreover, Multi-way RE is also known as record extraction. The focus of this work will be on binary relationships. In binary RE, relationships are extracted from natural language text, typically from a sentence, in the form of triples. The triples can be defined as relationship instances of the form $(ent_1, ent_2, rel)$, where $ent_1$ and $ent_2$ are named-entities and $rel$ is the relationship between them, e.g., *(Barack Obama, Honolulu, birthPlace)* extracted from the sentence *"Barack Hussein Obama II, né le*

*4 août 1961 à Honolulu (Hawaï), est un homme d'État américain"*, which roughly translates to *"Barack Hussein Obama II, born August 4, 1961 in Honolulu (Hawaii), is an American statesman"*. Moreover, RE can be seen as a multi-class classification problem, because the goal can be described as predicting a relationship type, out of multiple types, i.e., classes, for each entity pair.

In Section 3.2.1, relevant features for RE are presented. Moreover, in Section 3.2.2 existing techniques for RE, relevant to the scope of this thesis are detailed. Furthermore, in Section 3.2.3, several software tools that are able to perform RE are described.

### 3.2.1 Features

The following features are useful for RE [3]:

- *Surface Tokens:* tokens between and around the entities. They are useful because they contain clues for the relationship held between the entities
- *POS tags:* important because verbs are good indicators of the relationship between entities (that tend to be nouns or noun phrases)
- *Parse Tree:* more useful than just POS tags, because it shows how the words are related
- *Dependency Graph:* is just as adequate as a parse tree and less expensive to create

It is also possible to extract a *dependency path* from a dependency graph. A dependency path between two nouns in the graph, i.e., between the two named-entities in the sentence, consists on the path obtained by traversing the dependency graph and obtaining the sequence of words and/or their classes following the orientation of the graph from one entity to the other.

### 3.2.2 Techniques

There are various types of techniques than can used to perform RE [11]:

1. **Rule-based methods**, where patterns are manually created to extract the relationship between two entities.
2. **Supervised methods**, that require labeled training data to train conventional classifiers to predict the relationship between two entities in a sentence. They can be divided into feature-based, kernel-based and neural-based.
3. **Semi-Supervised methods**, also referred to as bootstrapping (and sometimes weakly supervised). These methods use a small amount of seed relationship instances or seed patterns, to iteratively extract more relationship instances and patterns from a large amount of unlabeled data. They are explained in Section 3.2.2.1.

4. **Distantly Supervised methods**, where an existing database of relationships, i.e., a knowledge base, is used to get relationship instances. These instances are used to automatically label sentences where the respective named-entities occur together, that are then used to train a classifier. These techniques are detailed in Section 3.2.2.2.

5. **Unsupervised methods**, also referred to as Open Information Extraction. Relationships are extracted without labelled data or set of pre-defined relationships. It is often targeted for doing RE over the web. Moreover, relationships are sets of strings (typically containing a verb) that are mapped to canonical forms. More detail on this type of techniques is given in Section 3.2.2.3.

In the following Sections we will be focusing on the RE techniques that do not require manually labeled data or a high amount of linguist knowledge, which are the most relevant and that fit the scope of this thesis, i.e., semi-supervised, distantly supervised and unsupervised methods.

#### 3.2.2.1 Semi-Supervised Methods

Supervised methods require a large amount of labeled data, which usually is not available. Moreover, manually labeling enough examples to be able to train a classifier is expensive. To solve this, semi-supervised methods use a limited amount of labeled data and a large amount of unlabeled data.

Semi-supervised RE, through the use of an iterative bootstrapping process, assumes that we have a few seed triple instances of the target relationship(s) and a large unlabeled text corpus. For example, if the target relationship is "birthPlace", some seed entity tuples could be *(Barack Obama, Honululu)*, *(Albert Einstein, Ulm)*, *(Kate Hudson, Los Angeles)*. A bootstrapping procedure will extract similar entity pairs that have the same target relationship, e.g., *(Elvis Presley, Mississippi)*. Moreover, a bootstrapping procedure [11] takes the seed triple instances as input and looks for sentences, in the unlabeled text corpus, where the entities occur together. Then, it takes the context of the sentences, e.g., words surroding entities, and generates patterns that are then used to extract new triple instances. The process repeats until reaching a stopping criteria that can be, for example: an acceptable number of triple instances, the number of new triple instances generated in an iteration is too small or no new triple instances are learned. It is also possible to start the process by using seed patterns. A phenomena called semantic drift can commonly occur in these type of approaches: when a wrong pattern extracts wrong instances which will lead to the generation of wrong patterns, making the extracted instances "drift".

DIPRE [44] (Dual Iterative Pattern Relation Extraction) was one of the first bootstrapping techniques to be proposed. This technique makes use of the *pattern-relation duality*: good patterns extract good instances, thus good instances generate good patterns. In the experiments reported by the authors, they have the goal of extracting author-book relationship instances from web documents. As initial seeds, they used five pairs of (author, book) instances, e.g., *(Charles Dickens, Great Expectations)*.

The DIPRE algorithm starts by finds all occurrences of the seed instance tuples *($ent_1$, $ent_2$)* in the web documents. It collects the order in which $ent_1$ and $ent_2$ occur in the sentence, as well as their context: the URL of the web document and the surrounding words i.e., the words before the entities (prefix), the words in between (middle), and the words following the entities (suffix). Then, patterns are generated and defined by: $(i)$ the order of the entities, $(ii)$ the prefix of the URL page and $(iii)$ the surrounding words (prefix, middle and suffix), i.e., *(order, url-prefix, prefix, middle, suffix)*. Patterns are used to find and extract more pairs of instances that are added to the initial seeds, and the process repeats. A pair *($ent_1$, $ent_2$)* matches a pattern if it has the same order as the pattern, the URL of the page matches *url-prefix* and it contains a text segment that matches the following expression, depending on the order: **prefix, $ent_1$, middle, $ent_2$, suffix* or *prefix, $ent_2$, middle, $ent_1$, suffix**, e.g., "*Book* was written by *Author*", "*Book* by *Author*", etc.

An heuristic is used to avoid semantic drifting, by making sure that patterns are not too general, because that means instances that do not correspond to the target relationship may be extracted. The heuristic consists in measuring the specificity of the pattern and rejecting patterns with a low specificity. The specificity of a pattern $p$ is defined as: $specificity(p) = |p.urlprefix||p.prefix||p.middle||p.suffix|$, where $|x|$ corresponds to the length of $x$. Moreover, a pattern $p$ is valid if $specificity(p) * n$ is above a certain threshold $t$, where $n$ corresponds to the number of occurrences that match pattern $p$.

Snowball [45] is based on the DIPRE approach and introduces new improvements, namely a new strategy for pattern generation and tuple extraction and strategies for scoring patterns and tuples.

One of the key differences between DIPRE and Snowball is that Snowball patterns include the target named-entity types: with the pattern "*Organization* is located in *Location*", any pair of entities with the words "is located in" between them would match the pattern when using DIPRE. However, with Snowball, only entities that are of the target type, in this case, *Organization* and *Location*, would match the pattern. To achieve this behavior, it is necessary to label the text being analyzed with named-entities, using NER.

Patterns are represented by a tuple composed of the named-entity types, and the left, middle and right contexts (i.e., surrounding words of the entities like the prefix, middle and suffix in DIPRE): *(left, $type_1$, middle, $type_2$, right)*, where $type_1$ and $type_2$ correspond to the named-entity types of the first and second occurring entities. These contexts are represented by weight vectors, i.e., a vector associating weights to the words in the given context. Typically, the middle context gives more clues regarding the relationship between two entities, so its vector weights will have higher values than the ones from the left or right context vectors. In fact, using the weight vector approach, patterns are more flexible and minor variations on occurrence's contexts will still match a pattern. To measure the similarity between patterns or instance tuples it is necessary to compute the sum of the inner products between each context vector.

Snowball then uses the patterns to find new instance tuples: it finds all occurrences of the target named-entity types in the corpus e.g., *Organization* and *Location*. Next, it collects their contexts and

transforms them into vectors. Additionally, it measures the similarity between each occurrence tuple and all existing patterns. A tuple is added to the initial set of instanece seeds if the similarity between it and at least one pattern is higher than a given pre-defined threshold.

The most relevant improvement of Snowball over DIPRE is the pattern and tuple evaluation that enables to control the problem of semantic drift. Snowball measures the *confidence* of each pattern it generates and only keeps patterns with high confidence, so it does not consider patterns that will most probably extract wrong tuples. The same is done in regards to tuples, because wrong tuples may generate inappropriate patterns, which in turn will extract more wrong tuples.

Some of the following works [46] include [47] that uses bootstrapping to automatically learn surface patterns for Question-Answer (QA) systems. The answers for some question types, which express relationships, e.g., *"birthPlace"*, with questions like *"Where was Person born?"*, follow patterns, e.g., *"Person was born in Location"*. They propose a bootstrapping approach to automatically learn the patterns starting from a few QA seed pairs. Furthermore, [48] propose Espresso, a bootstrapping system to learn relationships such as *"partOf"* or *"isA"*. It uses generic patterns and filters incorrect instances, making use of the pattern learning algorithm of [47]. Moreover, [49] explore the use co-reference information (all expressions that refer to a given entity in the text) to improve bootstrapping results, in particular the recall. And, [50] focus on exploring different word cluster features for RE, where words that occur in similar contexts are grouped in the same cluster.

Although the different approaches try to solve semantic drift, the problem still ocurs. Moreover, bootstrapping requires having a set of seed instances for each relationship type one wants to extract, and those can influence the performance of the algorithm.

#### 3.2.2.2 Distantly Supervised Methods

Distant supervision [51] was proposed as an alternative paradigm to RE that does not require labeled training data, which is expensive to produce. It uses the idea of bootstrapping, by similarly using seed relationship instances. However, instead of using a small amount of seed instances or patterns to start the process, it uses a large database that contains relationships, i.e., knowledge base (KB), e.g., Freebase [52], now Wikidata [53], DBPedia [7], Yago [54]. The assumption (known as the *distant supervision assumption*) is that, *if a pair of named-entities in the KB hold a certain relationship, then any sentence that contains both entities is probably expressing that relationship*. Following the assumption, for every triple $(ent_1, ent_2, rel)$ in the KB, sentences containing the entity pair are collected from a large unlabeled text corpus, from the same domain as the KB. Consider the example in [11], where we have a KB of relationships. For the triple *(Albert Einstein, Ulm, birthPlace)*, sentences like *"Einstein was born in Ulm"*, *"Einstein, born (1879), Ulm"*, *"Einstein's birthplace in Ulm"* will be extracted from the corpus.

Furthermore, features are extracted from all the sentences that contain the entities, and used to

create training instances to train a classifier. Additionally, the process requires NER to, first, identify the named-entities e.g., Person, Organization and Location, in every sentence of the corpus. Negative training instances i.e., instances expressing the nonexistence of a relationship between two entities, are also necessary. These are created by randomly taking entity pairs from the KB that do not occur together in a relationship, and applying the same procedure.

The approach proposed by [51] used Freebase as the KB, and collected sentences from Wikipedia articles. They used a multi-class logistic regression classifier, that receives as input a pair of entities and a respective feature vector. The features extracted from all the sentences, for a given entity pair, are aggregated in a single feature vector. Different features were extracted from each sentence using a conjunction of lexical features, e.g., surface words and their part-of-speech tags, and a conjunction of syntactic features, e.g, dependency path between the two entities in the sentence, plus the named-entity classes of the entities. They came to the conclusion that using a combination of both lexical and syntactic features yields the best performance. Considering the sentence *"Scientist Einstein was born in Ulm"*, an example lexical conjunctive feature obtained from the sentence could be:

*<Left = [Scientist], E1 = Person, Middle = [was/VERB born/VERB in/ADP], E2 = Location, Right = []>*

Although this paradigm allows the automatic creation of training data, following the distant supervision assumption generates noisy instances because:

1. not all sentences containing both entities express the relationship in the KB, which leads to the creation of false positive instances. For example, consider the relationship triple *(Barack Obama, Honolulu, birthPlace)* and the sentence *"Obama returned to Honolulu to live with his maternal grandparents"*. Although it contains both entities in the triple, the relationship expressed in the sentence between them is different.

2. if the KB is incomplete, i.e., does not contain all entity combinations for a given relationship, when creating negative instances, false negative instances will be generated. For example, if two entities are related in "the real world" but a triple that relates them is not present in the KB, they may be used for generating negative instances.

3. two entities may hold more than one relationship between them, meaning that a sentence containing both entities may not express the relationship from the current triple, from the KB, being processed. For example, if a Person was born and died at the same Location, then, there will be, at least, two relationships between the two entities.

When using a KB, e.g., Freebase like [51, 55], where the relationship instances are derived from the text corpus used for training, e.g., Wikipedia, it is easy to align the instances to the text. Moreover, the

work of [56] states that the noise generated by the distant supervision assumption is even more evident when using text not directly related to the KB, a harder and more real scenario. The extraction of new relationships will not always be from text from which the training KB is derived from. In both cases, and more evidently in the latter, named-entities may occur in the same sentence and the sentence not express the relationship between them. To fix this problem, they propose to relax the distant supervision assumption, creating the *expressed-at-least-once assumption*, that says: *if a pair of named-entities is related in the KB, at least one sentence containing both entities might express the relationship*. The distant supervision problem becomes a multi-instance learning (MIL) problem, where the sentences are grouped in bags, for each entity pair, that, according to the assumption, will contain at least one positive example for their relationship. Furthermore, this means that the RE stops being on sentence-level, where a relationship is predicted for each input sentence, to start being on a bag-level, where a relationship is predicted for each entity pair or bag. They train a graphical model on data created by applying their proposals to the New York Times (NYT) corpus, using Freebase as the KB. This was when the considered "standard" distant supervision NYT corpus (for English) was created.

The previous model is multi-instance single-label, thus does not capture that the same entity pair may have more than one relationship between them. Moreover, [57] present MultiR, a probabilistic graphical model of MIL that allows a pair of entities to have multiple labels, i.e., more than one relationship, outdoing the previous model. MIML-RE [58] improves by proposing a multi-instance multi-label graphical model model that jointly models, for a pair of entities, all instances and all their labels, i.e., relationships.

Most features of these models are derived from NLP tools, that introduce errors and noise into the models. [59] introduced deep learning with multi-instance learning to distant supervision using Piecewise Convolutional Neural Networks (PCNN), inspired by the work of [60] for supervised RE, to automatically learn the representation of instances, thus avoiding the need to design and engineer features that resort to NLP tools. In RE, labels are not predicted for each word in an input sentence, instead, all local features need to be considered in order to make a global prediction and the convolution is a way to naturally join those features. A Convolutional Neural Network (CNN) is a type of neural network specialized for a grid of values such as a matrix composed of stacked word representations, e.g., word embeddings, of the words that make up a sentence. The main layer of a CNN is the convolutional layer in which a convolution operation is applied. A convolution is a mathematical linear operation where the multiplication of a set weights with the input is performed. A Piecewise CNN, i.e., PCNN, is a variant of a CNN with an added piecewise max pooling layer, that has the goal of capturing structural information between the two named-entities expressing a relationship in the input sentence. Furthermore, they showed that using a PCNN is more beneficial that using CNNs.

Moreover, [61] propose a sentence-level attention-based model. A CNN (or PCNN) is used to construct a sentence representation, and, then, to alleviate the noise caused by the noisy instances,

sentence-level *attention* is used to select, from all sentence instances of a given entity pair, the ones that actually express the relationship. They show that PCNN with selective attention over instances (PCNN-ATT) performs better than CNN-ATT.

RESIDE [62] propose the use of side information that can be obtained from KBs, to softly impose constraints when predicting relationships. It makes principled use of $(i)$ entity types, because the types of the entities involved in a relationship are constraint by it, and $(ii)$ relationship aliases, which are various terms for expressing a given relationship. It also uses Graph Convolution Networks (GCNs) to encode syntactic information and improves the performance even without the use of side information. [63] consider both intra-bag and iter-bag attention to improve upon the noise that exists in distant supervision data. Some of the most recent state-of-the-art work, like HRERE [64], propose to unify the learning of RE and KB embeddings (KBE) to improve RE itself. KBE is task whose goal is to represent the KB entities and relationships in a vector space.

#### 3.2.2.3 Unsupervised Methods

The methods presented in the previous sections only extract relationships that are in a a pre-defined set of target relationship types. Moreover, the techniques either need a corpus annotated with entities and relationships, hand-crafted rules, seed relationship instances, or a database of relationships, i.e., KB. Either way, data annotated in some form is required; only the relationships annotated in the corpus, in the rules or seeds, or present in the KB are extracted. These approaches are not able to discover new relationships and thought and effort is necessary for each relationship type one wants to extract.

Unsupervised methods, also denoted as Open Information Extraction (Open IE) [65] methods, do not require any annotated data nor a defined set of target relationships. Instead, an Open IE technique, automatically extracts all relationship types it is capable of finding in a text. Additionally, these methods have the goal of making the extraction domain-independent, while being efficient and fast at extracting relationships from large domain-heterogeneous corpus i.e., the Web. A generic way of expressing relationships between two entities, commonly referred to as arguments in Open IE, is needed; [65] proposes the identification of relationship phrases which are phrases that typically correspond to relationships. We will be following [66] to introduce some works in this field.

Open IE was introduced with TextRunner [65], which implements a self-supervised learning technique. A Naive Bayes classifier is trained to predict if a sentence expresses a relationship (positive example) or not (negative example). To train the classifier, a small set of sentences was labeled as positive or negative using heuristics based on dependencies generated by a parser. The sentences are represented using shallow features, i.e., lightweight, e.g., POS tags. Then, the classifier is used during extraction as the first step, where only positive sentences are kept and, from those, candidate triples are created by identifying the relationship phrase and pairs of noun phrases (NP) in each sentence. In [67]

the same approach was applied, however a CRF model is trained instead of a Naive Bayes classifier. The CRF model learns to identify the spans of words that likely express the relationship between the entities.

The Wikipedia-based Open Extractor (WOE) [68] also uses a self-supervised learning approach. WOE obtains its labeled sentences by heuristically matching attribute-value pairs in Wikipedia's articles infoboxes[9] to sentences in the article. They experiment with training a CRF on shallow features like [67] and another on features from dependency-parse trees, which allow capturing long-range dependencies. The latter, at the cost of speed, increases the results when compared to using shallow features.

The previous two systems made use of automatically learned data; ReVerb [69] is an Open IE system that improves upon the previous approaches using hand-crafted rules. It proposes the implementation of a syntactic and a lexical constraint for extraction of relationships expressed by verbs. ReVerb syntactically constraints the relationship phrase (by means of a regular expression), using POS tags, to be the longest sequence starting with a verb, optionally followed by a preposition or nouns adjectives, adverbs, pronouns or determinants and a preposition. A lexical constraint is also applied in order to eliminate long and rare relationship phrases. Only relationships that occur a $k$ number of times with different arguments (i.e., entities) are considered. The nearest NP to the left and right of the relationship phrase are the arguments of the relationship.

OLLIE [70] is the first Open IE system that extracts relationships not solely mediated by verbs, like ReVerb, but also mediated by nouns, adjectives and other structures. It joins the ideas of using automatically learned data with ReVerb. A set of high-precision ReVerb relationship triples is used to bootstrap sentences based on them. Then, it uses the sentences to learn a set of extraction patterns based on the dependency tree of the sentences. Sometimes, relationships are only valid under a certain condition or attribution, i.e., are not factual, which is also expressed in the sentence where the relationship was identified. With ReVerb, relationships that are not factual are often extracted because only a local analysis of the sentence is performed. OLLIE adds context information to the extracted relationships, regarding if and which condition or attribution they are under. OLLIE's patterns are derived from ReVerb seeds, so, although it is able to extract noun relationships, the coverage on them is limited. Moreover, ReNoun [71] is an Open IE system that complements previous systems by focusing on extracting noun-based relationships and extracts many relationships that do not exist in OLLIE.

KrakeN [72] is the first system built for the purpose of n-ary relationships. It uses hand-crafted rules based on dependency parse trees to obtain the relationship phrases and arguments. EXEMPLAR [73] uses a similar approach but in addition uses the ideas of Semantic Role Labeling (SRL), to assign to the arguments of the sentence a semantic role, e.g., agent, instrument, etc. OLLIE's successor, OpenIE4 [74], combines two systems: SrlIE [75], that uses SRL using the verb as the relationship phrase

[9] https://en.wikipedia.org/wiki/Help:Infobox

and the role labeled arguments as the entities, and RelNoun [76] that extracts noun-based relationships, incorporating knowledge about compound relational nouns and demonyms [10].

The use of *contextual sentence decomposition* (CSD) is explored in the CSD-IE system [77]. For a given sentence, CSD will determine its contexts, i.e., sub-sequences that "belong together" in a semantic way. Each context contains a relationship that may depend on the other contexts, thus the idea of nested relationships is introduced. NestIE [78] more recently pick up on the idea of nested relationships.

More recent approaches are clause-based, where triple relationships are extracted from individual, independent and simplified sentence clauses. Clauses are groups of words, that are part of a sentence, and express some information. They can contain a subject, a verb, direct and indirect objects, complements and adverbials. ClausIE [79] uses dependency parsing to identify the set of clauses in each sentence. Then, it tries to identify their type based on its constituents. Finally, for each clause, based on its type, the relationship triple or triples are generated to represent the different pieces of information.

Stanford Open Information Extraction (Stanford OpenIE) [80] shifts the focus from building a large set of extraction patterns to build a classifier that is able to extract clauses from longer sentences. The system starts by splitting the input sentence into a set of independent clauses using the classifier. Afterwards, each clause is maximally shortened, i.e., minimized, while maintaining the necessary context, to produce shorter sentence fragments, that are easy to segment into relationship triples using patterns.

MinIE [81] is a recent Open IE system built over ClausIE, which they state, produces extractions that are too specific. Just like Stanford OpenIE, it tries to minimize both the relationship phrase and arguments, by removing unnecessary and too specific parts. It also annotates each extracted relationship with semantic annotations, like OLLIE, with context information about polarity, modality, attribution and quantities. Graphene [82] is another recent Open IE system that starts by simplifying the complex structure of sentences, removing unnecessary parts. From this simplified structure it is able to extract meaningful n-ary relationships. These are enriched with contextual information and semantic links are added to connect them. Also recently, [83] assume a supervised learning approach to Open IE. They formulate Open IE as sequence labeling problem and train a bi-LSTM model to predict a label for each token in a sentence (entity, argument or other), using a modified variant of the QA-SRL dataset [84]. Moreover, [85] propose a neural Open IE approach, that does not resort to NLP tools that introduce errors in the systems. A enconder-decoder framework is used, where the encoder takes the input sequence and encodes it in a context vector, used by the decoder to create the labeled output sequence. The train data is obtained by giving OpenIE4 sentences from Wikipedia to extract relationship triples.

The Figure 3.6, adapted from [66], shows the extractions from different Open IE systems for the sentence: *"If he wins five key states, Republican candidate Mitt Romney will be elected President in 2008"*. It is possible to get an ideia of the kind of extractions produced by different Open IE systems.

[10] Demonyms are word expressions derived from locations to denote the residents of that location, e.g., French, Japanese, etc

```
OLLIE:
(1) (Republican candidate Mitt Romney; will be elected President in; 2008)
    [enabler=If he wins five key states]
(2) (Republican candidate Mitt Romney; will be elected; President)
    [enabler=If he wins five key states]
(3) (Mitt Romney; be candidate of; Republican)
(4) (Mitt Romney; be candidate for; Republican)
(5) (he; wins; five key states)
ReVerb:
(6) (he; wins; five key states)
(7) (Republican candidate Mitt Romney; will be elected President in; 2008)
ClausIE:
(8) (he; wins; five key states)
(9) (Republican candidate Mitt Romney; will be elected; President in 2008 If he wins
five key states)
(10) (Republican candidate Mitt Romney; will be elected; President in 2008)
Graphene:
(11) #1 CORE (Mitt Romney; will be elected; President)
("a) CONTEXT:NOUN BASED Mitt Romney was a republican candidate .
("b) CONTEXT:TEMPORAL in 2008 .
("c) CONTEXT:CONDITION #3
("d) CONTEXT:NOUN BASED #2
(12) #2 CORE (Mitt Romney; was; a republican candidate)
(13) #3 CONTEXT (he; wins; five key states)
```

**Figure 3.6:** Open IE systems' extractions for the sentence *"If he wins five key states, Republican candidate Mitt Romney will be elected President in 2008"*

### 3.2.3 Tools

There are several software tools available for performing RE. These tools can be divided in the following types: $(i)$ black-box web services, $(ii)$ Open IE systems and $(iii)$ third-party libraries that distribute pre-trained models and also allow the users to train their own model. Most of the RE tools available work exclusively for English. This Section describes the tools that we consider the most relevant.

Other tools we found include Open Calais which is a black-box web service, that we already described in Section 3.1.3, capable of performing RE for English. We also found more Open IE systems and black-box web services, that we will not be including in this Section.

We found a third-party library, Stanford Relation Extractor[11], that implements a supervised RE model [86] (for English), included in Stanford's CoreNLP [39] set of tools. It also allows training new models by providing labeled data. jSRE[12] is another tool that implements a supervised machine learning technique that allows one to train, again, by providing labeled data.

[11] https://nlp.stanford.edu/software/relationExtractor.html
[12] https://hlt-nlp.fbk.eu/technologies/jsre

#### 3.2.3.1 Stanford OpenIE

Stanford Open Information Extraction (Stanford OpenIE)[13] is a Java software implementation of an Open IE system for English whose algorithm [80] was described in Section 3.2.2.3. Its most recent version is available through Stanford CoreNLP.

For each sentence, the system returns relationship triples along with the confidence of the extraction *(confidence, subject, relation, object)*, where the subject and object are the named-entities.

#### 3.2.3.2 TextRazor

TextRazor[14] is a black-box web service that performs several NLP tasks using machine learning combined with several repositories, e.g., with named-entities, to parse, analyze and extract semantic meta-data from text. It is capable of performing NER and RE, altough the latter, only for English.

To use the TextRazor REST API, you simply use an key to make an HTTP Post request to the API. TextRazor also offer SDKs for Python, PHP and Java, built on top of their API. Moreover, the free plan allows $500$ requests per day and at maximum $2$ concurrent requests.

#### 3.2.3.3 IBM Watson NLU

IBM Watson Natural Language Understanding (IBM Watson NLU) is a web service, already described in Section 3.1.3, as a tool capable of extracting meta-data from text including named-entities and relationships between them. In particular for French, and some other languages, the system is capable of extracting several relationship types such as[15]: bornAt, diedAt, locatedAt, ownerOf, residesIn, spouseOf.

For each relationship detected, the system returns the sentence that contains the relationship, the relationship type, the named-entities involved and a confidence score for the relationship. As with other black-box web services, it has limitations, particulary on the free version.

#### 3.2.3.4 ReVerb

ReVerb is an Open IE system, whose algorithm was described in Section 3.2.2.3. The KnowItAll group at the University of Washington has developed and released a software implementation of ReVerb, and is available for download[16]. It is an Open IE tool, which extracts binary relationships from English sentences at a Web-scale i.e., large scale. It is implemented in Java and can be executed from the command line or included in a Java project.

[13] https://nlp.stanford.edu/software/openie.html
[14] https://www.textrazor.com/
[15] https://cloud.ibm.com/docs/natural-language-understanding?topic=natural-language-understanding-relation-types-version-2
[16] http://reverb.cs.washington.edu/

It receives as input a file composed of sentences. For each sentence given as input, ReVerb outputs a tuple $(argument_1, relationship\ phrase, argument_2)$, i.e., the two named-entities and the relationship between them. It can also obtain and output the confidence score it has on each extraction. Other metada about the extraction can also be obtained e.g., POS tags of the sentence's words.

We also found a French adaptation of ReVerb [5], developed by the RALI team at the Université de Montréal and that is available in GitHub[17]. RALI modified ReVerb, to use French statiscal models, instead of the English ones used by ReVerb, and changed the regular expression used for extracting the relationship phrase to work for French. Contrary to the original ReVerb system, the French adapation is not able to provide a confidence score for the extractions.

#### 3.2.3.5 OLLIE

OLLIE was also developed by the KnowItAll group, and is available for download[18]. The algorithm was described in Section 3.2.2.3. Like ReVerb, it is an Open IE tool for extracting binary relationships from English sentences. Moreover, it is implemented in Java and can be executed from the command line or added to a Java Project.

The system receives as input a text file where each line is a unique sentence. For each sentence, it returns a tuple, $(argument_1, relationship\ phrase, argument_2)$. However, it is also capable of capturing and outputing enabling conditions and attribution clauses. An enabling condition is a condition that the extraction depends on to be true e.g., "if he wins five key states" in the sentence "If he wins five key states, Republican candidate Mitt Romney will be elected President in 2008" in Figure 3.6. An attribution clause states the entity that asserted the extraction and the verb in that context e.g., "early astromners" and "believe" in the sentence “Early astronomers believed that the earth is the center of the universe”.

#### 3.2.3.6 OpenNRE

OpenNRE [6] is an extensible toolkit that provides a framework to implement Neural RE (NRE) models. It is soon to be deployed as Python package[19]. It includes steps of data processing, model training and experimental evaluation. English pre-trained models are available.

It allows training typical NRE models for sentence-level, bag-level, and others, although, it is specially tailored for the supervised sentence-level scenario. Furthermore, its extensibility allows for an easy implementation of new RE models, using OpenNRE. In particular for bag-level models, it allows training a CNN or a PCNN with or without attention. Moreover, [59] showed that PCNN perform better than CNN and, [61] showed that a PCNN with attention, i.e., PCNN-ATT, suprasses a CNN with attention, i.e., CNN-ATT, as we detailed in Section 3.2.2.2.

[17] https://github.com/rali-udem/reverb-french
[18] https://knowitall.github.io/ollie/
[19] https://github.com/thunlp/OpenNRE

OpenNRE allows to easily make experiments and research different model settings. They also provide typical metrics for evaluation of the models in the different RE scenarios. Additionally, it facilitates the deployment of the models to be used in "real-world" applications.

#### 3.2.3.7 Discussion

| | Black-box | Open IE system | Third-party library | French support |
|---|---|---|---|---|
| Stanford OpenIE | | x | | |
| TextRazor | x | | | |
| Open Calais | x | | | |
| IBM Watson NLU | x | | | x |
| ReVerb | | x | | x |
| OLLIE | | x | | |
| OpenNRE | | | x | |

**Table 3.2:** RE tools

Table 3.2 summarizes the most relevant RE tools. We only found two tools capable of performing French RE *off-the-shelf*: IBM Watson NLU and the French adaptation of ReVerb. The first is a black-box web service and the latter an Open IE system.

Moreover, TextRazor and Open Calais are other black-box tools that are only capable of dealing with English texts.

Stanford OpenIE, ReVerb, OLLIE are all Open IE systems. Except for ReVerb which has an adaption for French, the systems are only capable of dealing with English texts.

OpenNRE was the only third-party library that we found, that besides allowing to train supervised machine learning models, i.e., sentence-level training, also implement bag-level training, a widely used setting for distant supervision. The techniques it implements for RE are neural-based.

# 4

# Creating a French NER model

## Contents

The goal of this Chapter is to describe the approach that we took for developing a Named-Entity Recognition (NER) solution to integrate CONNECTIONLENS.

The approach for performing NER may consist in using a software tool or implementing a technique from scratch. Due to the availability of a great amount of software tools that can work for French, there is no need to implement a technique from scratch. Moreover, software tools can be distinguished in two forms: $(i)$ black-box web services or $(ii)$ third-party libraries.

*Black-box* tools typically support multiple languages and many include French. They also do not require having labeled data. However, they impose constraints that are not desired for CONNECTIONLENS. *Third-party libraries* provide pre-trained machine-learning models or enable to train a model. In general, they have pre-trained models, at least, for the English language. However, SpaCy and Flair are the only tools that have French pre-trained NER models available.

Since we found several French datasets annotated with named-entities, we decided to train our own French NER supervised machine-learning model using two third-party libraries, i.e., SpaCy and Flair, to see if we can improve upon their pre-trained models. We chose these tools because they use recent deep-learning based state-of-the-art techniques. We created a distinct model for each tool and selected the one that performs better.

The remainder of this Chapter is organized as follows: Section 4.1 presents the pipeline for creating our French NER model. Moreover, Section 4.2 describes the datasets used to train the models. The pre-processing of each dataset, their combination and their posterior division in sets, is described in detail in Section 4.3. Additionally in Section 4.3, each dataset is explored, so we can learn about their characteristics. Models are evaluated throughout the Chapter, and the evaluation methodology and metrics used are explained in Section 4.4. To obtain the best possible model for each tool, we perform model selection, which is detailed in Section 4.5. Each tool's selected model is evaluated using the test set, in Section 4.6. Finally, in Section 4.7, according to the results of the previous Section, a new model is created using the complete dataset.

## 4.1 Pipeline

The pipeline that enables the creation of our NER model is represented in Figure 4.1. The pipeline begins by subjecting each of the datasets (i.e, *Quaero Old Press*, *KB Europeana Newspapers* and *WikiNER*) to a *pre-processing step*. This pre-processing step comprises several sub-steps that may include mapping to another encoding scheme, conversion to another format, tokenization, etc. Then, the datasets are combined to form a *combined dataset*, with the goal of providing the models with as much information as possible for learning. The *combined dataset* is divided randomly, in terms of sentences, in a train, development and test set.

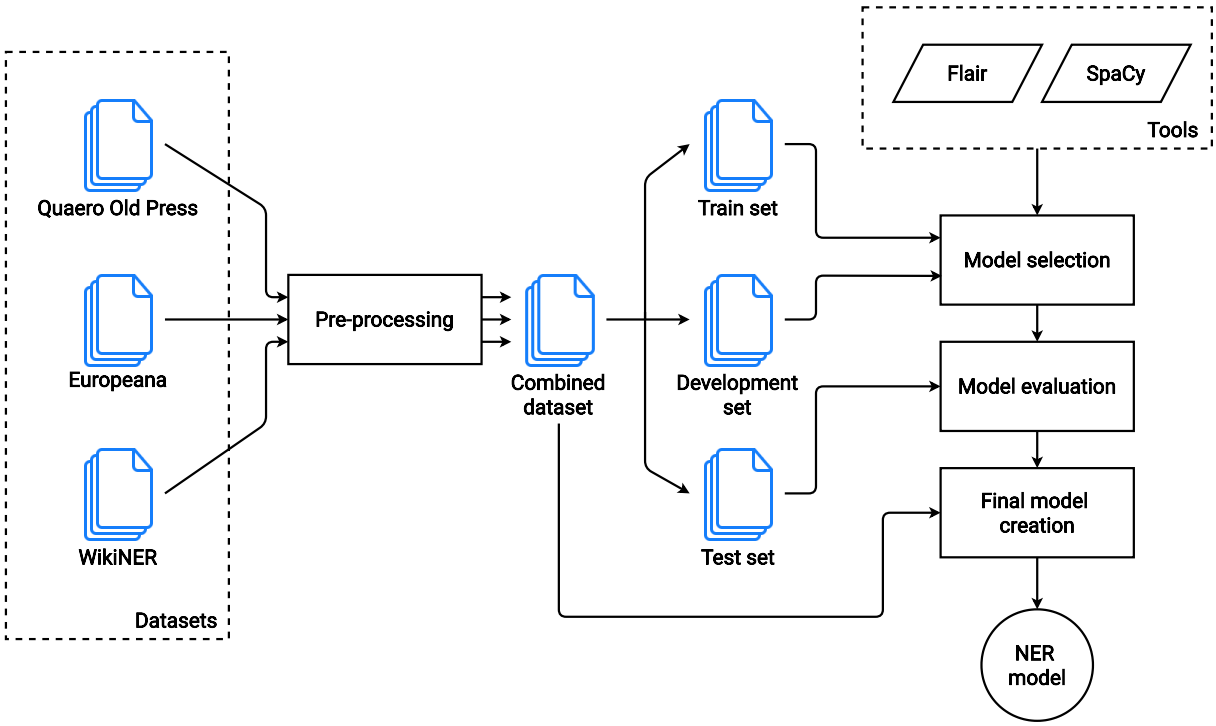


**Figure 4.1:** Model creation pipeline

The train and development sets are used in the *model selection step*. Model selection (also called hyperparameter selection, optimization or tuning) consists in finding the best performing model out of a set of models obtained by different hyperparameter configurations. The train set is used to train different models that are the result of different hyperparameter configurations. The model, i.e., hyperparameter configuration, that yields the best results on the development set is selected. Model selection is performed for each tool, thus we obtain the best model possible for each tool.

Each tool's selected model is evaluated using the test set, in the *model evaluation step*. The goal of model evaluation is to estimate the model's generalization performance i.e., how it performs on unseen data, using the test set. According to the results, we choose the best performing model out of the two, and create our NER model by training, again, on the complete dataset, in the *final model creation step*.

Afterwards, our NER model will be evaluated against the pre-trained models that we found and against the current model in CONNECTIONLENS. The best performing model will be integrated in CONNECTIONLENS.

## 4.2 Datasets

In this Section, we describe the datasets, annotated with named-entities, that we used and combined for creating the NER models.

### 4.2.1 WikiNER

WikiNER [41] is a labeled multilingual dataset for NER, automatically created using the text and structure of Wikipedia[1]. Two datasets are available for each language, wp2 and wp3. The datasets are labeled with *Person*, *Organization*, *Location* and *Miscellaneous* named-entities. The labels follow the IOB encoding scheme, also referred to as IOB-1. A pipe-delimited format is used, in which there is one sentence per line, with each token information separated by a whitespace. The token information consists of the token itself, its POS tag and its NER label, that are separated by a pipe character, also referred to as vertical bar. An example of a line of this dataset is:

$$France|NAM|I-LOC \quad dans|PRP|O \quad (..)$$

In what concerns the text, Punkt [87] was used to split paragraphs into sentences and a Penn Treebank-style [88] tokenizer, modified to accommodate the different languages, for tokenization. Furthermore, hyphenated terms were kept as single tokens. In what concerns personal titles, these were removed from *Person* named-entities for English and German, however there is no mention in [41] in what regards French. After manually accessing the dataset, we conclude they are part of the named-entity.

The wp3 dataset contains more sentences than wp2, so we decided to use wp3. Regarding the French language dataset, the wp3 version contains around $3.5M$ tokens, where $134K$ sentences were selected from about $15K$ articles.

### 4.2.2 Europeana

The KB Europeana Newspapers NER dataset (which we refer to in the document as simply Europeana) was developed in the Europeana Newspapers project[2]. In summary, the goal of the project was to improve access to digitized newspaper collections. The collection in question has more than $1000$ digitized newspapers, published between 1618 and 1990, from $23$ European libraries and $40$ languages.

The digitized images of the newspapers were put through an OCR[3] (Optical Character Recognition) process, creating fully searchable text versions.

Training datasets and models for NER were produced for Dutch, French and German using a subset of the original collection [89] with the goal of facilitating work on NER in the context of historic newspapers. Specifically for French, whose newspapers were provided by the National Library of France, there were $207,000$ tokens, with $5,672$ *Person*, $5,614$ *Location* and $2,574$ *Organization* named-entities.

[1] https://www.wikipedia.org
[2] http://www.europeana-newspapers.eu
[3] OCR (also sometimes referred to as text recognition) is the process of recognizing text in images and converting it into machine-readable text.

Moreover, as a result of the texts being the product of OCR, there is a tendency for errors to occur during the recognition of the characters. Furthermore, not all named-entities in the text are annotated, due to annotators' errors.

Although it is announced that the labels of all datasets use a BIO encoding scheme, i.e., IOB-2, that is not true for the French dataset. An IO encoding scheme is used, where problems arise when two named-entities of the same type occur immediately after each other.

The NER datasets and models are available in KB Research Lab[4] (original versions) and in GitHub[5], where work is still being done to improve the quality and harmony of the datasets: adding metadata (e.g., source article), correcting OCR errors, removing erroneous sentences, removing hyphenation[6], adding or correcting NER labels, etc. In both versions there is no representation of sentences (usually delimited by empty lines), although sentences exist in the dataset's text. There is also no mention on how tokenization was performed.

Focusing on the GitHub version, a version of the French dataset where the IO encoding scheme was converted to BIO, i.e., IOB-2, as been made available. A CoNLL style format is used, where each line represents a token and its information is in tab-separated columns: the first column corresponds to the position of the token in the sentence, the second column to the token itself, the third column to the NER label and the fourth column to an embedded NER label. This embedded NER label corresponds to another named-entity that might occur simultaneously, e.g., University of Paris is an *Organization* with an embedded *Location*. None of the different languages' datasets contain embedded entities yet, but the column is already included. An example of a line in this dataset is:

$$0 \qquad Paris \qquad B-LOC \qquad O$$

### 4.2.3 Quaero

The Quaero Old Press Extended Named Entity dataset [90], or corpus (which we refer to in the document as simply Quaero), is distributed by the European Language Resources Association (ELRA)[7].

The dataset is composed of $76$ newspaper issues, provided by the National Library of France in digitized images and in OCR format, and, published between 1890 and 1891, by the French newspapers "Le Temps", "La Croix" and "Le Figaro". We confirmed that these newspaper issues are not the same as the ones in the Europeana dataset (described in Section 4.2.2).

From the images and the OCR output, $295$ pages were extracted in text format and were manually annotated with named-entities. The Quaero dataset is divided in a train dataset and a test dataset. The

[4] http://lab.kbresearch.nl/static/html/eunews.html
[5] https://github.com/EuropeanaNewspapers/ner-corpora
[6] Hyphenation is an automatic process where words are broken in two separate lines by means of an hyphen character that would otherwise go further than the right margin of a page.
[7] http://catalog.elra.info/en-us/repository/browse/ELRA-S0349

train dataset contains $231$ pages with $1,297,742$ words, $114,599$ types and $136,113$ components. The test dataset contains $64$ pages with $363,455$ words, $33,083$ types and $40,432$ components. There is also a sub-dataset that is supposed to work as a mini-reference dataset, for quality evaluation purposes.

For both the train and test files, there is a normalized version, which is recommended by the authors, although, no information is provided as to what differs between the raw and the normalized version.

The dataset is composed by two kinds of elements: types (and sub-types), that refer to the category of a named-entity, and components, that categorize the elements inside a named-entity.

There are $7$ types of named-entities and $32$ sub-types, that include classical named-entities: *Person*, e.g., individual person (pers.ind), *Location*, e.g., physical location (loc.phys.geo), *Organization*, e.g., administration (org.adm) and additional named-entities: *Time*, *Amount*, *Production* and *Functions*.

Components can be $(i)$ *transverse*, which means they can be used for any named-entity type, e.g., name of the entity, qualifying adjective, number, and $(ii)$ *specific*, which means they are specific for a certain type of named-entities, e.g., first/middle/last name (for the sub-type *Individual Person*), week/-day/month/year (for the *Date* type).

Additionally, annotations have a non-flat structure and it is possible to define three kinds of composition that can take place: $(i)$ a type contains a component, $(ii)$ a type includes another type (used as component), and $(iii)$ a named-entity type is used to refer to another named-entity type, for when the type of the entity is different in the context it is inserted into.

In terms of the format of the dataset, it consists in having the text with XML tags marking the occurrence of the elements, i.e., types and components. Below is an excerpt of the dataset as an example:

*00212633/PAG_1_TB000042.png*
*D'abord faisons trêve aux dissentiments*
*politiques. Quand la Foi est en péril, redirons-*
*nous avec <pers.ind> <name> Léon </name> <qualifier> XIII </*
*qualifier> </pers.ind> , tous doivent s'unir d'un*
*commun accord pour la défendre.*

Metadata is placed multiple times on a file, and it identifies a block of text that was selected for annotation, e.g., 00212633/PAG_1_TB000042.png.

It is important to note that the dataset contains noisy and incorrectly recognized characters since it is a result of an OCR operation. And, since the images belong to newspapers, which are written as columns, this results in text with line breaks and hyphenation, as it is possible to see in the excerpt given as an example. As a result of these characteristics, some features were added:

- An attribute "correction", used only on named-entities whose text is erroneous (and not on non-entities), as a result of the OCR operation. The erroneous text is kept but its correct form is inserted

in a correction attribute placed in the type element, i.e., tag. For example (taken from [90]):

*<pers.ind correction="Le Moine">*
*<name.last> LE Moibte. </name.last>*
*</pers.ind>*

- A component "noisy-entities" used for OCR errors involving a named-entity boundary. It enables annotating an entity that is in a span of noisy characters. For example (taken from [90]):

*<loc.adm.reg correction="EN ALSACE-LORRAINE">*
*<noisy-entities> KN_ALSACE'LOBR4INE </noisy-entities>*
*</loc.adm.reg>*

General principles were defined to assist the human annotators, for example, punctuation next to a named-entity should be kept that way and annotated as part of the named-entity, e.g., *<loc.adm.town> Paris. </loc.adm.town>*. These principles are described in detail in [90].

## 4.3 Pre-processing

The goal of the pre-processing step is to uniformize all datasets to have the same format, encoding scheme and named-entity types.

Regarding the named-entity types, it is necessary to select a common set to all the datasets. After inspecting the datasets, the common set of named-entity types corresponds to the traditional set: *Person*, *Location* and *Organization*. Moreover, these are represented, respectively, by: "PER", "LOC" and "ORG". Each have an associated prefix, resulting from the encoding scheme being used, e.g., "B-PER".

An IOB encoding scheme, more specifically, IOB-1, was chosen as it is widely used, and the improvement obtained from using more expressive encoding schemes, e.g., IOBES, is usually not significant (see [32, 91]). Furthermore, with this encoding scheme we benefit from using the popular "conlleval" evaluation script made available by several CoNLL shared tasks (e.g., CoNLL-2002[8]).

A CoNLL style format was chosen, where the dataset file contains $2$ columns separated by a whitespace. Moreover, there is a token per line and empty lines identifying sentence boundaries. Each line has the token itself separated by a whitespace from its NER label. For example:

$$Italie \quad I-LOC$$

Besides, obviously, benefiting from the use of the "conlleval" evaluation script by using a CoNLL style format, this format is also accepted, or easily convertible to be acceptable, by both tools used to train the models, i.e., Flair and SpaCy.

[8] https://www.clips.uantwerpen.be/conll2002/ner/

The pre-processing step corresponding to each dataset is detailed in Sections 4.3.1, 4.3.2 and 4.3.3. After the pre-processing, a data exploration step was performed for each pre-processed dataset, with the goal of learning their characteristics, and is described in Section 4.3.4. Moreover, the joining of the datasets to create a combined version is described in Section 4.3.5, and the following division in train, development and test sets is explained in Section 4.3.6.

### 4.3.1 WikiNER

The WikiNER dataset already uses the target encoding scheme, i.e., IOB-1. However, in contrast, it has a different format, i.e., pipe-delimited, so it was necessary to convert it to the target format. Additionally, it has an extra named-entity type that was removed, i.e., *Miscellaneous*, to fit the chosen named-entity types set *{ Person, Location, Organization }*.

Moreover, the wp3 version of the dataset was chosen over the wp2 version, because it contains more sentences, as explained in Section 4.2.1.

### 4.3.2 Europeana

The Europeana dataset uses a similar CoNLL style to the target, yet, it has extra columns with information that is not necessary. Therefore, those were removed and only the token and its NER label were kept.

Since it is impossible to automatically convert from an IO to an IOB encoding scheme, it was necessary to use the GitHub version of the dataset, that uses IOB-2, which we converted to IOB-1, the target encoding scheme. For this, it was necessary to replace all "B" prefixes with "I", except on the first token of a named-entity that follows a named-entity of the same type.

Furthermore, the named-entity types of the dataset correspond to the target set.

### 4.3.3 Quaero

Since the goal is to combine all three datasets previously described, it did not make sense to keep the train and test divisions in the Quaero dataset. Therefore, all train and test files were transformed to the target specifications (that we defined at the beginning of the section) and concatenated, making up a single file.

To transform the Quaero dataset to have the target format, encoding scheme and named-entity types, it was necessary to: $(i)$ remove and collect the named-entities tags from the text, $(ii)$ perform tokenization on the tag free text, and $(iii)$ establish correspondences between each token and the named-entities collected.

The named-entities present in this dataset have a wide range of types and are also sub-divided in sub-types. It was necessary to create a mapping between the Quaero sub-types and the named-entity target, i.e., the *{ Person, Location, Organization }* set:

- All location types were kept despite of the sub-type, i.e., "loc\..*", and mapped to "LOC"
- All organization types were kept despite of the subtype, i.e., "org\..*", and mapped to "ORG"
- The sub-type *individual person* was kept, i.e., "pers.ind" and mapped to "PER". The sub-type "pers.coll" was not kept because it translates to a group or collectivity of persons, e.g., catholiques, which does not correspond to the meaning associated with *Person*.

To apply the transformation steps described before, it was mandatory to pre-process the text first (strongly based on the use of regular expressions):

1. Remove metadata.
2. Remove empty lines.
3. Remove hyphenation. As a result of newspapers being written in columns, due to space, words are often hyphenated.
4. Separate tokens attached to tags. This consists in finding characters immediately followed by an angle bracket "<", or preceded by an angle bracket ">", and placing a space between them. For example *" (...) </pers.ind>UVAL "* becomes *" (...) </pers.ind> UVAL "*.
5. Apply corrections. This consists in finding all correction attributes and replacing everything inside the corresponding tag by the correction text. For example *" <title correction="Saint-Père"> sâi*nt-Père </title> "* becomes *" <title> Saint-Père </title> "*.

   Moreover, not all correction attributes were correctly placed by the annotators, meaning that their automatic application will result in the removal of text that should not be removed. For example:

   *<org.adm correction="Commission">*
   *<kind> Commissioji </kind>*
   *du*
   *<name>budget</name>*
   *</org.adm>*

   would become:

   *<org.adm>*
   *Commission*

*</org.adm>*

The correction attribute should have been placed in the component "kind", to correct the actual erroneous word.

6. Remove angle brackets that are not part of tags. An example of an angle bracket that is not part of a tag is *" P<*che extraordinaire "*.

7. Remove erroneous hyphens. If an hyphen is not joining two word parts, e.g., "Saint-Père", then we considered it an error of the OCR process and remove it.

8. Remove extra whitespace characters and whitespace from the beginning of sentences.

After the pre-processing was performed, we split the text into tokens, based on whitespace. We removed from the tokens the tags that were not of interest (and corresponding component and noisy-entities tags), while keeping the text they enclosed. Afterwards, we transversed the list of tokens and assigned each of them a non-entity "O" label, except when we found a tag, which we removed from the tokens and assigned the tokens it enclosed the corresponding label based on the tag type. Furthermore, only the most exterior tag was considered, e.g.,

*<org.ent>*
*<kind>l'école</kind>*
*de*
*<loc.adm.town><name>Paris</name></loc.adm.town>*
*</org.ent>*

All tokens inside the sub-type "org.ent" tag would be assigned the ORG named-entity, despite there being another sub-type tag inside of this tag.

A first version of a CoNLL IOB-1 format was built, by creating a token column and a label column obtained from the previous process. Since proper tokenization was not performed, this first version is incorrect in terms of token definition, e.g., elisions like "l'école" are considered as one token when it should be "l'" and "école" or words are attached to a comma, e.g., "gouvernement,". And, as a result, it is also wrong in terms of encoding. For example:

*Paris, I-LOC*
*Rome B-LOC*
*et B-LOC*

The token "Rome" is assigned a label with "B" prefix, because it immediately follows a named-entity of the same type, "Paris,". However, if properly tokenized, punctuation would not be part of the tokens

(except in special cases like abreviations) and "Rome" would be assigned the same label but with "I" prefix, because it would come after the token "," which has the "O" label:

*Paris I-LOC*
*, O*
*Rome I-LOC*
*et O*

In order to correct this, we took the pre-processed text and removed all tags. Then, we tokenized and segmented the text into sentences. Keeping in mind the goal of uniformizing the datasets, we want to take a similar approach for sentence segmentation and tokenization to the other datasets (that are already tokenized). We have no information in regards to the Europeana dataset, however we know that the WikiNER dataset used Punkt for sentence segmentation and a Penn Treebank-style tokenizer. We decided to use the Stanford tokenizer[9], which is a Penn Treebank-style based tokenizer, for which there is a derivative French version, with rules for elision and compounding. It is available in the Stanford CoreNLP Java distribution, but also in Python, through wrappers, e.g., stanfordnlp.

Since, we are implementing in Python, it made sense to use a Python wrapper, therefore we decided to use stanfordnlp. We made sure that hyphenated words were kept the same and that no token was represented by more than one word or had whitespace. The tokenizer introduces some errors in the dataset, however it is not something unexpected, as the other datasets also have tokenization and segmentation errors. Furthermore, along with the tokenization, sentence segmentation was also performed by stanfordnlp, and empty lines were added to create boundaries between sentences.

We transverse the "incorrect" token list, with the "erroneous" labels, and the "correct token" list, and establish correspondences between the tokens, assigning a correct label to the correctly tokenized tokens.

### 4.3.4 Data Exploration

In this section we explore each of the datasets' characteristics.

We are interested in learning about attributes, like the number of sentences, tokens and named-entities in each dataset. These attributes are represented in Table 4.1. Each row corresponds to a dataset and each column to a different attribute.

We can observe that, WikiNER is the largest dataset and Europeana is the smallest. There is no value assigned to the number of sentences in the Europeana dataset, because, as explained in Section 4.2.2, there is no definition of sentences in the dataset.

Looking at the named-entity distribution over each dataset, shown in Figures 4.2, 4.3 and 4.4, it is possible to observe the percentage of named-entities belonging to each type.

[9] https://nlp.stanford.edu/software/tokenizer.shtml

| | #sentences | #tokens | #named-entities |
|---|---|---|---|
| WikiNER | 132,257 | 3,499,695 | 216,341 |
| Europeana | - | 205,914 | 9,894 |
| Quaero | 72,083 | 1,851,076 | 75,500 |

**Table 4.1:** Dataset's attributes

Moreover, organizations are consistently the least represented in the datasets, with less than $25\%$ percent of the named-entities belonging to this named-entity type.

There is always a discrepancy between the representation of each named-entity type, except between locations and persons in the Europeana dataset, where the percentage of named-entities is roughly the same. In the WikiNER dataset, more than $50\%$ of its named-entities are locations, being the most represented named-entity type on this dataset, as opposed to the Quaero dataset, where the majority of the named-entities are persons.

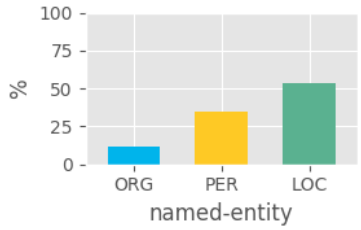


**Figure 4.2:** WikiNER's named-entity distribution

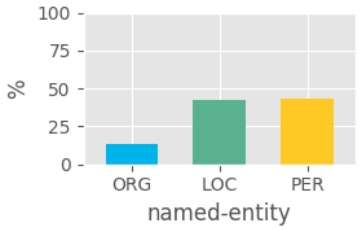


**Figure 4.3:** Europeana's named-entity distribution

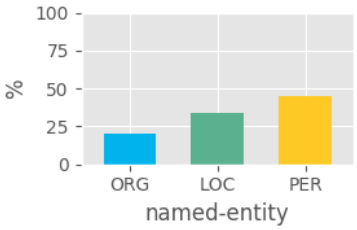


**Figure 4.4:** Quaero's named-entity distribution

### 4.3.5 Combining the Datasets

The goal was to combine the three pre-processed datasets into a single dataset. However, both the Europeana and the WikiNER dataset showed some undesirable aspects.

The Europeana dataset has no representation of sentences, as explained in Section 4.2.2. It would be necessary for us to perfom sentence segmentation, because, in the next step, the train, development and test sets will be obtained by dividing the combined dataset in terms of sentences. One solution would be to take all tokens, from the token column, in the CoNLL formated Europeana dataset, join them and use a sentence segmenter to perform sentence segmentation over this text. The difficulty with this solution lies in the fact that it is not possible to automatically know how to join the tokens: some tokens should be joined next to each other, e.g., "l'" and "école", and others with whitespace between them, e.g., "," and "Paris" or "les" and "autres". This solution seems feasible, however, it would be necessary to be aware of all the different joining conditions. Additionally, the dataset is not "perfect" and contains erroneous characters that are the result of the OCR operation. For example:

*à O*
*Di I-LOC*
*. I-LOC*
*nard I-LOC*
*, O*

A period appeared in the middle of the named-entity "Dinard" during the OCR operation, and this resulted in there being three tokens, "Di", "." and "nard" when there should only be one token "Dinard". Moreover, if we were to join the tokens, and considering that typically, a period would join next to its previous word and there would be a whitespace between the period and its following word, if applied here, would create "Di. nard" and induce similarly or more in error.

Another solution would be to go through the token column in the dataset and finding sentence-ending characters, i.e., period ".", question mark "?" or exclamation mark "!". However, the OCR errors would also induce in error. Taking the previous example, the period would be considered the end of a sentence when in reality is just a erroneous character.

Moreover, after the pre-processing for the dataset was performed, we noticed that there were a lot of errors in what regards the use of B prefixes (probably a result of the conversion from IO to IOB-2). Below are some examples (that use IOB-1 since they are from after we pre-process the dataset):

| | | |
|---|---|---|
| *Raoul I-PER* | *la O* | *Le O* |
| *Robin I-PER* | *rue I-LOC* | *Conseil I-ORG* |
| *, O* | *de B-LOC* | *de B-ORG* |
| *Marie I-PER* | *la B-LOC* | *l' B-ORG* |
| *Collin B-PER* | *Tannerie B-LOC* | *Ordre B-ORG* |

In all examples shown, the B prefixes should actually be I prefixes, because their corresponding tokens are not a new named-entity but rather the continuation of a named-entity.

More occurrences of errors of the same pattern, as in the examples, occur throughout the dataset. These errors, in contrast to other errors presented before, imply that the IOB-1 encoding scheme is wrong and would induce the models trained with this dataset in error. Solving these errors would require French linguistic knowledge to verify and and apply manual corrections to the dataset. For these reasons, we decided not to use the Europeana dataset.

In what regards the WikiNER dataset, we also found wrong uses of the B prefix that do not match to what we consider as a correct. Take these examples:

| | | |
|---|---|---|
| *) O* | *l' O* | *journaliste O* |
| *et O* | *Irlande I-LOC* | *chez O* |
| *Eugène I-PER* | *du B-LOC* | *Rolling I-ORG* |
| *Hugo B-PER* | *Nord I-LOC* | *Stone B-ORG* |

In these examples, all B prefixes should also be I prefixes, for the same reasons presented for the

Europeana examples. For this reason, we decided not to use the WikiNER dataset.

In conclusion, instead of combining the three datasets as originally planned, we decided to only use the Quaero dataset, due to the errors in the other datasets' encoding scheme.

### 4.3.6 Train, Development and Test Sets

The chosen dataset, needs to be divided in train, development and test sets, to be used by the tools that will produce the NER models i.e., Flair and SpaCy.

To achieve this, we took all sentences in the dataset and randomly divided them in three sets. This division is done percentually, and it is possible to choose what percentage each set should have of the original sentences. Considered by some as a rule of thumb, we decided to divide the dataset in $60\%$ for the train set, $20\%$ for the development set and $20\%$ for the test set.

## 4.4 Evaluation Methodology

When evaluating the extraction of a NER model or system, we care about how it predicts named-entities, and not each token, since that is the goal of this task. Additionally, we consider an exact-match evaluation, this means, both the boundaries and the named-entity type predicted need to match the true annotation in the dataset to be considered *correct*. This is the adopted evaluation procedure in multiple CoNLL shared tasks. Moreover, we use the "conlleval" evaluation script to measure the performance of each model.

It is important to also understand the following numbers, computed for each named-entity type:

- True positive (tp): number of correctly predicted named-entities
- False positive (fp): number of incorrectly predicted named-entities
- False negative (fn): number of named-entities not predicted

The metrics used to evaluate each model's performance are computed for each named-entity type and they are (following the definition given by CoNLL):

- **Precision:** which shows the proportion of correctly predicted named-entities of a given type out of all named-entities predicted as that type: $precision_{type} = \frac{tp}{tp+fp}$
- **Recall:** which shows, for a named-entity type, the proportion of named-entities that are of that type, that were predicted as that type $recall_{type} = \frac{tp}{tp+fn}$
- $F1$**-score:** which is the harmonic mean of precision and recall, so it gives an idea of how the model performs in terms of the both measures $F1 - score_{type} = 2 * \frac{precision_{type} * recall_{type}}{precision_{type} + recall_{type}}$

Moreover, we aim to maximize the $F1$-score.

Since this is a multi-class problem, precision, recall and $F1$-score are computed for each named-entity type, as well as their micro-average. While, for a given metric, the macro-average computes it for each named-entity type and then takes the average, the micro-average joins the contributions of every named-entity type to compute the average metric. This means, macro-average treats all named-entity types equally and micro-average treats all named-entity instances equally. CoNLL uses micro-averaging to obtain the overall metrics.

## 4.5 Model Selection

In this Section, we describe the *model selection step*, explaining how the best model possible was obtained, for each tool, i.e., Flair and SpaCy.

### 4.5.1 Flair

The Flair framework allows training sequence labelling models using a bi-LSTM-CRF architecture and facilitates the integration with different word embeddings. Additionally, it allows combining or stacking different word embeddings by concatenating their vectors.

It is stated that a hyperparameter selection routine is implemented, which allows defining a search space of hyperparameters, e.g., different numbers of RNN layers, mini-batch sizes, word embeddings, etc. However, after experimenting with this part of the framework, we came to the conclusion it does not work due to flaws on the Flair implementation. After discussing with a Flair developer, the recommended solution was to compare different word embeddings by training different models (a model for each word embedding) and selecting the best performing model out of all.

We looked into all word embeddings that can work for French and trained a model using the train and development set using each word embedding or a combination of them:

- FastText embeddings (classical static word-level embeddings): *standard*
- Stacked forward and backward Flair embeddings (contextual string embeddings): *stacked-flair*
- Stacked FastText and forward and backward Flair embeddings: *stacked-standard-flair*
- Stacked FastText and character embeddings: *stacked-standard-char*
- Byte Pair embeddings (word embeddings precomputed on the subword-level): *bytepair-fr* (French) and *bytepair-multi* (multilingual)
- CamemBERT embeddings (a Tasty French Language Model): *camembert*

- XLM-RoBERTa embeddings (multilingual language model): *xlm-roberta-base*

We consider the model trained with FastText embeddings, "standard" embeddings, as the baseline to which we compare the results of the other models.

The maximum number of epochs correspond to the maximum number of times the learning algorithm is exposed to the full train set, and each epoch results in a model. In the end, the best model corresponds to the model that gives the best $F1$-score on the development set.

During training, at the end of each epoch, the current model is evaluated using the development set. If the resulting $F1$-score does not improve for a certain number of epochs (called patience, and that by default is $3$), the learning rate is decreased by a certain amount (called anneal factor, and that by default is $0.5$). The training stops when it reaches the maximum number of epochs (that by default corresponds to $100$ epochs) or when the learning rate falls bellow a certain threshold (called learning annealing, and that by default is $0.0001$). This is called learning rate annealing, and in this case it is done against the development set's $F1$-score. As explained previously is Section 3.1.2.2, after data is fed into the network, the errors are back-propagated and the network's weights are adjusted. The weight adjustment amount is called learning rate. It is a hyperparameter that reflects the rate to which the model learns. Having a good learning rate means that it needs to be low enough so that optimal weights are achieved, but high enough so that it learns fast. Instead of having a fixed learning rate value, Flair uses learning rate annealing to have an adaptative learning rate, where it is adjusted according to the models performance on the development set.

We limited the number of maximum epochs to $30$ during the training of the different word embedding models. After the best performing word embedding model is selected and evaluated on the test set, detailed in Section 4.6, it will be retrained on the complete dataset and with an increased number of maximum epochs, explained in Section 4.7.

In Table 4.2 the evaluation results of each model on the development set are presented. What we consider to be the standard model has a better precision overall and also for each named-entity type when compared to the recall, this means that the model makes sure that the named-entities it finds are correct at the cost of not detecting every named-entity.

For all models, organizations always have lower scores when compared to persons and locations, which might be a result of the dataset having a lower representation of organizations. Moreover, the former always has a higher precision than recall.

We can observe that the models trained using CamemBERT (camembert table) and stacked FastText and Flair embeddings (stacked-standard-flair table) are the two best performing models. Although the CamemBERT model is slightly better overall than the other model, it is much slower making predictions and it required changing the source code of the Flair library in order to make it work. For this reason, we selected the stacked FastText and Flair embeddings model.

| standard | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 79.30% | 76.35% | 77.80% |
| ORG | 71.40% | 57.96% | 63.98% |
| PER | 78.21% | 76.35% | 77.27% |
| Overall | 77.38% | 72.57% | 74.90% |

| stacked-flair | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 79.30% | 85.06% | 82.08% |
| ORG | 72.71% | 62.95% | 67.48% |
| PER | 84.34% | 85.61% | 84.97% |
| Overall | 80.47% | 80.77% | 80.62% |

| stacked-standard-flair | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 82.99% | 86.22% | 84.57% |
| ORG | 74.47% | 62.73% | 68.10% |
| PER | 85.37% | 87.71% | 86.52% |
| Overall | 82.63% | 82.08% | 82.35% |

| stacked-standard-char | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 80.01% | 84.08% | 82.00% |
| ORG | 71.55% | 63.39% | 67.22% |
| PER | 81.76% | 83.97% | 82.85% |
| Overall | 79.30% | 79.78% | 79.54% |

| bytepair-fr | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 79.29% | 77.84% | 78.56% |
| ORG | 69.21% | 58.09% | 63.16% |
| PER | 81.13% | 80.48% | 80.80% |
| Overall | 78.35% | 74.99% | 76.63% |

| bytepair-multi | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 80.12% | 76.28% | 78.15% |
| ORG | 68.51% | 62.26% | 65.23% |
| PER | 79.85% | 82.14% | 80.98% |
| Overall | 77.77% | 76.08% | 76.92% |

| camembert | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 85.26% | 85.26% | 85.26% |
| ORG | 71.72% | 68.83% | 70.25% |
| PER | 85.72% | 87.04% | 86.37% |
| Overall | 82.80% | 82.70% | 82.75% |

| xlm-roberta-base | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 80.31% | 76.49% | 78.35% |
| ORG | 69.57% | 51.71% | 59.32% |
| PER | 78.27% | 86.23% | 82.06% |
| Overall | 77.58% | 75.86% | 76.71% |

**Table 4.2:** Flair's model selection results

### 4.5.2 SpaCy

To train a model using SpaCy it is possible by means of Python code or the command-line interface (CLI). After discussing with a SpaCy developer, it was recommended to train using the CLI, as it is done for all of SpaCy's distributed models. They also referred that SpaCy does not have any built-in tools for hyperparameter selection and it is also, currently, not as easy as it should be to modify the model hyperparameters. Furthermore, all of SpaCy's distributed models were trained with the default hyperparameters. For this reason, we decided to train using the CLI and to not perform hyperparameter selection. Instead we experiment with pre-training SpaCy's "token to vector" layer by providing it with raw French data. The resulting pre-trained weights are provided to train a model instead of the convolutional neural network layers (refer to Section 3.1.3) being initialized with random weights. Additionally, we update SpaCy's pre-trained French NER model with our train set. Besides that, a standard model was also trained, i.e., a default model with no add-ons, that we consider as the baseline to which we compare the results of the other SpaCy models.

As with Flair, we limited the number of maximum epochs to $30$ and a model is saved after each epoch. Moreover, at each epoch the model being trained is evaluated using the development set, and the best model is the one that gives the best $F1$-score on the development set. Unlike Flair, by default there is no early stopping, i.e., the training does not end after a certain amount of epochs without improvment in the $F1$-score. Furthermore, SpaCy shuffles the train set at each epoch, so the order of examples shown to the model do not influence its learning.

SpaCy requires train data in JSON format, and there is a built-in CLI command that allows the conversion of different dataset formats to the JSON format, including CoNLL style formats, which is the format of our dataset. The conversion of both the train and development sets to the JSON format were the first steps. During this conversion, the encoding scheme is also converted to BILUO. So it was necessary to convert it back to BIO when evaluating the model.

The pre-trained weights were obtained by running a SpaCy CLI command, that as explained before, pre-trains the "token to vector" layer. More specifically, it trains the convolutional neural network to predict word vectors and then obtains the networks "pre-trained" weights. These pre-trained weights are loaded back during initialization of the convolutional neural network when training the NER model. To train this layer it is necessary to provide it with raw text for it to learn to predict a word's vector, taking into account its surrounding words. This raw text needs to be in a JSONL (newline-delimited JSON) format to be accepted by SpaCy, which consists in, and specifically in our case, in having sentences represented by a JSON object, per line, whose value is a list of the tokens that form the sentence, for example:

*{"tokens": ["Mais", "ce", "sont", "ses", "propres", "vêtements", "qui", "ont", "pris", "feu", "."]}*
*{"tokens": ["Le", "retour", "de", "flamme", "ne", "s'", "est", "pas", "fait", "attendre", "."]}*

The raw text was obtained from a dataset of crawled French news articles, from the top $10,000$ news sites, made available by Webhose.io[10]. This dataset is composed of $245,308$ JSON files. It was necessary to pre-process these files in order to convert the raw text contained in them to the JSONL format, which includes steps of segmentation and tokenization of the text (using the approach taken for the Quaero dataset in Section 4.3.3). This resulted in a total of $3,457,619$ sentences and $85,359,724$ tokens, which we reduced to a total of $70,000$ sentences and $1,731,215$ tokens. We performed the "pre-training" for the default number of $1,000$ epochs, and took the weights of the last resulting model, i.e., the model that corresponds to the $1,000th$ epoch, and used them to train a NER model.

We chose to experiment with SpaCy's capability of online learning on their models to update one of their existing models, by training it with our train set, and seeing if there is a performance improvement. We decided to do this solely on SpaCy's medium model, instead of also doing it on the small model, because a bigger model is expected to perform better and small models do not include word vectors, which also impacts the performance. SpaCy's pre-trained models are better described in Chapter 6. The pre-trained models were trained on the WikiNER dataset, and thus are able to extract *Person*, *Location*, *Organization* and *Miscellaneous* named-entities. Our dataset does not include the *Miscellaneous* named-entity type, so it is ignored during the evaluation.

In Table 4.3, the evaluation results of each of the three models on the development set are presented.

| standard | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 79.20% | 83.29% | 81.19% |
| ORG | 67.43% | 69.33% | 68.37% |
| PER | 82.59% | 84.56% | 83.56% |
| Overall | 78.33% | 81.01% | 79.65% |

| with-pre-trained-weights | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 79.86% | 83.75% | 81.76% |
| ORG | 66.24% | 69.65% | 67.90% |
| PER | 82.44% | 85.98% | 84.17% |
| Overall | 78.23% | 81.88% | 80.01% |

| updated | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 80.96% | 83.50% | 82.21% |
| ORG | 68.21% | 67.62% | 67.91% |
| PER | 83.55% | 85.77% | 84.65% |
| Overall | 79.61% | 81.28% | 80.44% |

**Table 4.3:** SpaCy's model selection results

For all models, we can observe that they have a higher recall than precision, not only in terms of the overall score but also for each named-entity type, except for the updated model where the precision is slightly higher for organizations. A higher recall translates in the models trying to predict all the named-

[10] https://webhose.io/

entities at the cost of wrongly predicting named-entities. Moreover, organizations have lower scores than the other named-entity types, which might be a result of its lower representation in the dataset, and, persons have the highest scores.

Using pre-trained weights decreased almost insignificantly the overall precision and improved very slightly the overall recall and $F1$-score, when compared to the standard model. The updated model shows an increase in all metrics in comparison to the standard model. Moreover, it has the highest overall scores for all metrics except for the recall, when compared to the model with pre-trained weights.

In theory, we should select the updated model since it has the higher $F1$-score. However, selecting the updated model means we always have to ignore *Miscellaneous* named-entity predictions when evaluating. Still, the updated model is bigger, which generally means better results. On the other hand, selecting the model trained with pre-trained weights, means we do not have to do any post-processing and the $F1$-score difference between the models is not very high. Both have advantages and disadvantages, however we decided to select the updated model.

## 4.6 Model Evaluation

The selected best performing models of each tool are evaluated on the test set, to get an estimate of their performance on unseen data. These results are presented in Table 4.4.

| flair-stacked-standard-flair | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 82.80% | 85.77% | 84.26% |
| ORG | 74.44% | 62.32% | 67.84% |
| PER | 86.12% | 88.19% | 87.14% |
| Overall | 82.88% | 82.01% | 82.44% |

| spacy-updated | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 81.06% | 83.76% | 82.39% |
| ORG | 68.81% | 66.61% | 67.69% |
| PER | 83.66% | 86.61% | 85.11% |
| Overall | 79.84% | 81.50% | 80.66% |

**Table 4.4:** Model evaluation results

Although the SpaCy model has a better score for the organizations recall, the Flair model is superior in all other metrics, overall and for all named-entity types. For this reason, the Flair model is chosen to advance to the next step in the pipeline.

## 4.7 Final Model Creation

According to the previous step, the best model we achieved uses the Flair and stacked FastText and Flair embeddings. The final step of the model creation pipeline is to train the chosen model on the complete dataset. However, it is necessary to keep a portion of the dataset to be used as the development set during the final training of the model. For this reason, we randomly divided the dataset again, but in two sets: train and development set, keeping $80\%$ of the sentences for the train set and $20\%$ of the sentences for the development set. We also increased the number of maximum epochs to $150$ while training. The model created will be referred to as *flair-ssf-quaero* for the rest of this document.

In Chapter 6, this version of the model will be evaluated on another dataset, and its results will be compared to pre-trained models and the current CONNECTIONLENS model. This way we ensure that the evaluations are not biased, since if all models were evaluated on this test set, our trained model might perform better, since it was trained with a similar train set i.e., a portion of the same dataset. This way we achieve a more independent and non biased evaluation.

# 5

# Distantly Supervised French RE

## Contents

The goal of this Chapter is to describe the approach that we took for developing a Relationship Extraction (RE) solution to integrate CONNECTIONLENS. Just like with NER, the approach can either consist in using a software tool or implementing a technique from scratch.

There are no datasets annotated with entities and relationships for French. Therefore, we are limited to using software tools that can perform French RE *off-the-shelf* or that facilitate the implementation of RE, by using techniques that do not require manually labeled data, such as the ones presented in Section 3.2.3. There are three types of software tools: black-box, Open IE systems and third-party libraries.

We only found two tools that can perform RE for French *off-the-shelf* : IBM Watson NLU and the French version of ReVerb, that we presented in Section 3.2.3. IBM Watson NLU is a black-box web service and thus entails the typical limitations and constraints, for instance, a fixed number of accesses per time period. Those constraints are not desired for CONNECTIONLENS. Moreover, the French version of ReVerb is an Open IE system, which means we have no control over which relationships are extracted, which is equally not desired for extracting relationships in CONNECTIONLENS.

Regarding third-party libraries, we only found one that implements a technique that does not require manually labeled data, which is OpenNRE. It integrates training of bag-level RE models, which is a widely applied method for distantly supervised RE, that we explained in detail in Section 3.2.2.2. With a bag-level model we do not predict the relationship present in each input sentence, but rather the relationship between two entities based on all sentences where they co-occur. To train this model we need to use distant supervision to automatically build a dataset using a combination of relationship instances, from a knowledge base (KB), and sentences expressing the relationships, from a text corpus, as we explained in Section 3.2.2.2. We decided to use French DBpedia and Wikipedia articles, respectively.

This chapter is organized as follows. We first describe the use of DBpedia and Wikipedia in Section 5.1. Then, we outline the steps to build the distantly supervised RE model in Section 5.2, which are further detailed in Sections 5.3, 5.4, 5.5, and 5.6.

## 5.1 DBpedia and Wikipedia

Wikipedia is an online free multilingual encyclopedia collaboratively constructed by a community of volunteer editors. Wikipedia articles are not only composed of "free" text but also structured elements, namely, infoboxes, that provide factual information about an entity, that the article is about, in the form of a list of attributes and respective values.

DBpedia [7] is a free multilingual KB built from Wikipedia's structured information. The information extracted from Wikipedia is stored in the form of *(subject, predicate, object)* triples using the Resource Description Framework (RDF). Over the years, the quality of the data has been improved by *mapping-*

*based infobox extraction*, where mappings have been developed to relate the infoboxes to the DBpedia ontology (structure that describes classes, e.g., *person*, and properties, e.g., *birth place*). This solves the problem of editors using different attribute names to express the same concept in infoboxes and consequently the concept being represented by different terms in the ontology, by mapping synonym infobox attributes to a single property or class in the ontology.

The fact that DBpedia and Wikipedia are available in French and the fact that mappings that relate Wikipedia to DBpedia exist, motivate our choice of using both to build a distantly supervised French RE dataset, together with the particularity that Wikipedia sentences typically state facts (unlike other text domains). Additionally, there is an increased probability that the DBpedia relationships appear in the sentences in Wikipedia articles since DBpedia is built from Wikpedia.

The DBpedia data set can be accessed online via the DBpedia SPARQL (query language for RDF data) query endpoint[1]. Alternatively, a dataset dump can be downloaded and stored using, for example, a local triplestore, also referred to as RDF store, which is a type of database specifically built for storing and querying triples. Since, the DBpedia SPARQL endpoint has query restrictions, we decided to download the last officially released DBpedia version, which is based on a Wikipedia dump from October 2016[2], for French, and query it locally using the OpenLink Virtuoso triplestore[3].

## 5.2 Procedure

Following the distant supervision *expressed-at-least-once assumption*: *if a pair of named-entities is related in the KB, at least one sentence containing both entities might express the relationship*, detailed in Section 3.2.2.2. Therefore, given a triple, $(ent_1, ent_2, rel)$, in the KB, it is expected that at least one sentence across both entities, $ent_1$ and $ent_2$, Wikipedia articles expresses the relationship $rel$. The procedure we took to build the distantly supervised RE model, based on the works of [92–94], consists in:

1. Get all relationships between named-entities of the type *Person*, *Location* and *Organization* from DBPedia, and filter and group the relationships. In addition, generate negative relationship triples, which are triples where the two entities involved are not related. This is detailed in Section 5.3.

2. For each entity involved in a relationship, process the text in its Wikipedia article and keep as candidate sentences the ones that contain at least two named-entities. Additionally, obtain and create a set of surface forms (alternative names an entity can be mentioned as, in the text) for the named-entity. This is further described in Section 5.4.

[1] http://dbpedia.org/sparql
[2] https://wiki.dbpedia.org/develop/datasets/dbpedia-version-2016-10
[3] http://vos.openlinksw.com/

3. For every relationship triple, access the candidate sentences of each named-entity involved, $ent_1$ and $ent_2$. Afterwards, using the named-entities' surface forms, select the sentences that match both named-entities, $ent_1$ and $ent_2$. This step is further described in Section 5.5.

4. Use the selected sentences to train a RE model using OpenNRE, as described in Section 5.6.

## 5.3 Relationships

We are only interested in extracting relationships between entities, namely of the *Person*, *Location* and *Organization* types, which are the named-entity types the NER solution in CONNECTIONLENS is capable of extracting. Therefore, we query DBpedia to obtain all relationships that exist between named-entities of those types, and obtain around $1.25M$ relationship triple instances in total.

Since there is a significant number of relationships types and, furthermore, a significant number of relationship types that are similar, i.e., which coarsely have the same meaning, and where sentences expressing them are expected to be similar, we decided to select a smaller set of relationships types. This set is the result of selecting the most frequent relationships types and grouping the ones with a similar meaning. For example, *region*, *country* and other relationship types were grouped in a single relationship type denoted *locatedIn*, because coarsely all of them want to express where something is located, and all the sentences that were to be selected had a similar structure.

For example, the following two triples and corresponding sentences were grouped together in the same *locatedIn* relationship type:

*(Wyoming, États-Unis, country):* Le Wyoming (...) est un État de l'Ouest des États-Unis.

*(Nicaragua, Amérique centrale, region):* Le Nicaragua, (...) est un pays d'Amérique centrale.

In total, we end up with $29$ relationship types: *locatedIn, birthPlace, deathPlace, party, familyMember, parentOrganisation, bandMember, sportsTeam, studiedAt, childOrganisation, nearTo, owner, spouse, influencedBy, leader, affiliatedWith, mentorOf, residesIn, employedBy, foundedBy, mentoredBy, recordLabel, influenced, coach, architect, operator, keyPerson, created* and *knownFor*.

Moreover, we reduced the number of relationship triples for time and efficiency reasons. We have to process the article's text of every entity that participates in a relationship, in step 2 of the procedure (that was explained in Section 5.2). Therefore, it makes more sense to remove entities, because we reduce the amount of articles we have to process, and, consequently, relationship triples are also removed, because the relationships where they participate are not considered. We removed a percentage of each type of named-entity from the set of entities, except organizations, i.e., removed $70\%$ of locations and $50\%$ of persons. Organizations are the least common type of entity in the dataset, with a difference of more than 100K from persons and locations, so we decided to keep all organizations. Consequently, we

have around $43K$ persons, $36K$ locations and $17K$ organizations, making a total of around $96K$ entities. In total, we end up with about $116K$ relationship triples.

The dataset being built also needs to contain negative examples, i.e., sentences that express the nonexistence of a relationship between two entities. To achieve this, we combine entities in a relationship triple *($ent_1$, $ent_2$, $NA$)*, where $NA$ denotes a negative relationship, that are not related in DBpedia, i.e., do not have a triple containing both entities in the KB. Then, in the next step (detailed in Section 5.4) sentences containing both entities are collected as examples of $NA$ relationship, based on the triples generated. We had to overestimate the number of negative triples to generate, because if two entities are not related, there is a low chance they appear together in a sentence. Ultimately, we add around $92M$ negative relationship triples to the previous $116K$ relationship triples. And, the $NA$ relationship type is added to the previous $29$ relationship types, making a a total of $30$ relationship types.

Moreover, we randomly split the relationship instances in the dataset, in train, development and test sets, using the rule of thumb of $60\%$ for the train set, $20\%$ for the development set and $20\%$ for the test set. It is important that the dataset is divided in terms of relationship instances, and not sentences, because sentences for the same entity pair should be together in the same set and not spread across the different sets.

## 5.4 Obtaining Candidate Sentences

We define as *candidate sentences*, for a given entity, all sentences in the entity's Wikipedia article text that contain at least two named-entities. The set of steps required to collect all candidate sentences for all entities in the dataset are described below.

For each entity in the processed dataset, we access the entity's French Wikipedia article. To access the Wikipedia articles we use the DBPedia NIF Dataset [95], included in the DBPedia 2016 version, which makes available, among other things, the actual text of the articles without structures like tables, etc.

Each article's text is segmented in sentences and tokenized using the Stanford tokenizer in the French version, which we previously used for the same task in Chapter 4. Then, NER is performed over the sentences, using the best French NER model we selected in Section 6.1.4, to annotate *Person*, *Organization* and *Location* named-entities in the text. We keep as candidate sentences the ones that possess two or more named-entities, as well as the respective detected entities.

We considered that the "name" of an entity in DBPedia corresponds to its Wikipedia's article title. There are different ways an entity can be referred to in the text, i.e., surface forms, besides its "name", so simply using exact matching to "names" to discover to what DBpedia entities, the entities detected in the text correspond to, is not sufficient. For example, a *Person* named-entity may be referred to, not

only by what commonly is the title of its article, i.e., first and last name, but also by just its first name, or just its last name, full name, as well as others such as nicknames, e.g., *Barack Obama* whose article title is *Barack Obama* may be referred to in the text, besides the article title, by *Barack*, *Obama*, *Barack Hussein Obama*, etc. The same applies for other types of named-entities, for example *Organizations* which may be mentioned in the text by the organization's full name, its acronym, among others.

For this reason, we decided to explore how to obtain a set of surface forms for each entity. The Lexicalizations[4] dataset provides surfaces forms for each DBPedia resource. The dataset is built from three different DBpedia properties an entity can have:

- *Label* - which contains the entity's article titles, e.g., the *label* property for *Harvard University* in DBpedia has: Harvard University (en), Universidad de Harvard (es), Universidade Harvard (pt)
- *Disambiguation* - which includes the ambiguous surface forms that can be associated with the entity in question, e.g., the *disambiguation* property for *Barack Obama* contains (where dbr means Dbpedia resource): dbr:Obama_(disambiguation), dbr:Barack_(disambiguation), dbr:Bama
- *Redirects* - which are URIs that are alternative to the entity's article and include the entity's surface forms, misspellings and acronym, e.g., the *redirects* property for *Barack Obama* contains: dbr:Obama, dbr:Barack_Hussein_Obama, dbr:Barack_H._Obama, dbr:Borrack_Obama, etc

However, the Lexicalizations dataset is only available in English, and although it could work for person names, in what concerns locations and organizations, the corresponding names can be different in English and in French. Therefore, to create, for each entity, a set of surface forms, we decided to use a combination of:

- the article title (in French) with and without the text between brackets removed (used for disambiguation), e.g., Ontario (Californie) and Ontario.
- the entity's *redirect* property values.
- if the entity belongs to the *Person* type, we add the first and last name separately to the set of surface forms.

  For example, the article title and (the only) redirect of *Michelle Obama* are: *Michelle Obama* and *Michelle Robinson* (her maiden name); if the text refers to *Michelle Obama* as just *Michelle* it will not be considered a match to *Michelle Obama* because *Michelle* is not in the set of surface forms. Moreover, last name is also necessary, for example, *Cristiano Ronaldo*, does not have *Ronaldo* in its redirects however, in the text, *Cristiano Ronaldo* is refered to as just *Ronaldo*.

[4] https://wiki.dbpedia.org/lexicalizations

- if the entity is of the *Organization* type we build an acronym by joining the first letter of each word in its name, i.e., article title. This is performed because there is a possibility the redirects may not include one.

All the strings in the resulting surface forms set are then pre-processed to be lowercase, without punctuation, accents and without stop words. Moreover, when trying to match detected entities to the surface forms, the same pre-processing is done to the detected entities string.

## 5.5 Selecting Sentences

For each relationship triple *($ent_1$, $ent_2$, $rel$)*, in each set, i.e., train, development and test set, we access the candidate sentences of the named-entities involved, $ent_1$ and $ent_2$. For each candidate sentence, we take the detected entities and pre-process the corresponding string like we did for surface forms in Section 5.4, i.e., lowercase, without punctuation, accents and without stop words.

Moreover, for each detected entity, we apply an exact matching procedure between its pre-processed string and all the surfaces forms of each involved entity, $ent_1$ and $ent_2$. If there is at least one surface form of $ent_1$ or $ent_2$ that matches the detected entity, and both the detected entity and the involved entity are of the same type, i.e., *Person*, *Organization* or *Location*, we consider it a match.

Finally, if the candidate sentence has at least one detected entity that matches $ent_1$ and at least one detected entity that matches $ent_2$, we select the sentence as an example of the relationship $rel$ between $ent_1$ and $ent_2$.

| | # entities | # instances | # sentences |
|---|---|---|---|
| train | 38,064 | 41,536 | 59,625 |
| dev | 11,451 | 10,269 | 14,707 |
| test | 14,177 | 13,063 | 19,322 |
| total | 63,692 | 64,868 | 93,654 |

**Table 5.1:** RE dataset attributes

The number of sentences we collected for each set is summarized in Table 5.1. We collected $59,625$ sentences for the train set, $14,707$ sentences for the development set and $19,322$ sentences for the test set, for a total of $93,654$ sentences. Moreover, the number of relationship instances, and entities, at the end of the procedure is not the same as at the end of the first step (described in Section 5.3), because some relationship instances may not have any sentence where both entities co-occur, thus the relationship instance is not considered. This happens more commonly with negative instances, due to the low probability of the entities co-occurring in a sentence if they are not related.

In Appendix A, we present a complete and detailed statistic regarding the dataset that resulted from

the procedure described. Namely, it is possible to observe the number of entities that exist per type, the relationship types that were picked and how many relationship instances and sentences exist per relationship type.

## 5.6 Training

We use the OpenNRE package [6] to train a bag-level Piecewise CNN (PCNN) with selective attention over instances (PCNN-ATT) model, the same architecture as the model proposed in [61], described in Section 3.2.

OpenNRE requires the dataset to be divided in three sets to train a model i.e, train, development and test sets. The division of the dataset was described in Section 5.3. Each of these sets has a corresponding file containing its example sentences, which are the sentences we selected in Section 5.5. The file needs to have the following structure: one example sentence per line, represented by a dictionary. Furthermore, the dictionary contains the following key-value pairs:

- *text* - whose value is the sentence itself. The sentence is previously tokenized, as described in Section 5.4.
- *relation* - the relationship type expressed in the sentence
- *h* - the head entity, i.e., $ent_1$, whose value is a dictionary with the keys: *id, name, and pos*. The *id* is a unique identifier for the entity, for which we used the DBPedia resource URI of the entity. The *name* corresponds to how the entity is mentioned in the text. Moreover, the *pos* is the position of the entity in the sentence, character-wise
- *t* - the tail entity, i.e., $ent_2$, whose value is a dictionary with the same structure as the one of the head entity *h*

The following example illustrates a representation of an example sentence for the relationship triple *(Barack Obama, Honolulu, birthPlace)*:

```
{
"text":  "Barack Hussein Obama II , né le 4 août 1961 à Honolulu .",
"relation":  "birthPlace",
"h":  { "name":  "Barack Hussein Obama II",
       "id":  "Barack_Obama",
       "pos":  [0, 23] },
"t":  { "name":  "Honolulu",
       "id":  "Honolulu",
```

```
        "pos":  [46, 54] }
}
```


Furthermore, if a selected sentence has more than one occurrence for $ent_1$ or $ent_2$, only the first occurrence is considered.

# 6

# Experimental Evaluation

## Contents

This chapter presents the experimental evaluation that we performed for both NER and RE models as well as the integration of the best performing model of each task in CONNECTIONLENS. The experimental evaluation carried out for NER is detailed in Section 6.1 and for RE it is detailed in Section 6.2. Section 6.3 presents the integration of the best performing NER and RE models in CONNECTIONLENS.

## 6.1 Named-Entity Recognition

This Section presents the experimental evaluation for NER. More concretely, in Section 6.1.1 the different models that we evaluated are introduced. Section 6.1.2 describes the dataset used to evaluate the models, and the pre-processing that it underwent. In Section 6.1.3, the evaluation methodology and metrics are presented. Section 6.1.4 reports the results achieved with each model and compares them.

### 6.1.1 Evaluated Models

We evaluated and compared the performance of different NER models:

- *flair-ssf-quaero* - the model selected and trained in Chapter 4 with the *Quaero* dataset using Flair and stacked forward and backward French Flair embeddings with French fastText embeddings.
- *flair-pre-trained* - the French pre-trained Flair model. It is trained with the *WikiNER* dataset, and uses French character embeddings, trained on Wikipedia, and French fastText embeddings.
- *spacy-pre-trained-md* - the medium pre-trained French SpaCy model. It is trained with the *WikiNER* dataset using the SpaCy's architecure described in Section 3.1.3.
- *spacy-pre-trained-sm* - the small pre-trained French SpaCy model. Trained with the same dataset and architecture as the medium model. However, unlike the medium model, it does not include word vectors.
- *stanford-quaero* - the model previously integrated in CONNECTIONLENS [2], trained using *Stanford NER* [39, 40], on the *Quaero* dataset.

### 6.1.2 Evaluation Dataset

For evaluating the different models, we chose the FTBNER [96] dataset. This dataset corresponds to a version [97] of the French TreeBank (FTB) corpus [98], annotated with named-entities. It was used to evaluate the CamemBERT language model [99] on the NER task. TreeBanks[1] consists in corpora containing manually checked linguistic annotations, such as syntactic structure, which is commonly

[1] https://copticscriptorium.org/treebank.html

represented as a tree, hence the name Treebank. Additionally, TreeBanks can be enhanced with other linguistic information. The FTB corpus, in particular, is composed of sentences extracted from the French newspaper *Le Monde*, of different domains, e.g., politics and economy, that span between 1989 and 1993.

The FTBNER dataset is a result of a manual annotation of named-entities and their respective types applied to the FTB corpus. Since it was manually annotated, it contains errors that typically derive from manual annotations, e.g., missing annotations, as also occurs with the datasets described in Chapter 4. Moreover, it contains $12,351$ sentences out of which $5,890$ contain at least one annotated named-entity. The dataset is labelled with $7$ different named-entity types: $2,025$ *Person*, $3,761$ *Location*, $2,381$ *Organization*, $3,357$ *Company*, $15$ *POI* (Point of Interest), $67$ *Product*, and $29$ *Fictional Character*. Following the annotation guidelines [96], *Fictional Character* is used for fictional people or animals, e.g., Zorro. *POI* is used for entities like stadiums, neighborhoods, ports, etc. *Organization* is used for all references to organizations except company names, where *Company* is used. In what concerns *Person*, used for individual persons, personal titles are not part of the named-entity as well as adjectives, determinants or professions.

The dataset is divided in three sets: train, developed and test. We noticed that the test set does not include example sentences for all named-entity types, e.g., *POI*. In what concerns the format, it uses a CoNLL style format, containing four tab-separated columns, where each line represents a token and empty lines define sentence boundaries. The first column corresponds to the token itself, the second column contains, for all tokens, an underscore, which might be the slot for some other information in the future, the third column corresponds to the POS tag and the last column to the NER label. Furthermore, the labels follow the BIO encoding scheme, i.e., IOB-2.

Since the models being compared are all trained using different datasets, by having an evaluation dataset not used by any of the models creates a better comparison in the sense that the dataset does not influence the results, e.g., if we compared all models using the test set used in Section 4.6, the models trained using the Quaero dataset could be positively influenced. By using FTBNER, we aim to achieve a more accurate evaluation, challenging all models by evaluating them on a completely new dataset, where the style of the text somewhat changes.

#### 6.1.2.1 Dataset Pre-processing

FTBNER was pre-processed in order to produce the named-entity types, encoding scheme and format as described in Section 4.3. The first step was to join the sets that FTBNER comes divided in into one dataset, with the goal of evaluating as much data as possible and have all named-entity types well represented.

The named-entity types used in this dataset were converted to the set *{ Person, Location, Organi-*

*zation }*, because it is common to the entity types that the models being evaluated were trained with. This set is represented, respectively, by the abbreviations: "PER", "LOC" and "ORG". It was necessary to remove *Product* and *Fictional Character* annotations as they do not fit the goal set. *Company* was converted to *Organization* and, after manual revision, *POI* was converted also to *Organization*. Furthermore, in FTBNER, the named-entity types are represented by their full name, so it was necessary to map them to the abbreviations, i.e., *Person* to "PER", *Location* to "LOC" and *Organization* to "ORG".

Like the previous evaluation of NER models, described in Chapter 4, we want to use the "conlleval" evaluation script, so, having the dataset in a CoNLL style format, with a column for the tokens and a column for their respective labels, is required. IOB encoding, more specifically IOB-1 also makes sense since the models that will be evaluated with FTBNER will predict the labels in that encoding.

Regarding the format, the dataset is in a CoNLL style similar to the target, so it was only necessary to remove the columns that contained non-necessary information, keeping only the token and respective NER label. Moreover, we converted the IOB-2 encoding scheme to IOB-1, the goal encoding scheme.

The dataset contains tokens that are composed by multiple words separated by underscore, e.g., "Il_est_vrai_que". Furthermore, hyphenated words contain underscores around the hyphen, e.g., "assurance_-_maladie". To match the other datasets, underscores were removed from hyphenated words and multiple word tokens were separated into multiple tokens with the appropriate label.

Both datasets used to train the models that will be evaluated, Quaero and WikiNER, generally, include personal titles as part of the named-entities, however, FTBNER does not. Since the evaluation consists in exact matching, if the model predicts the personal title as part of a given named-entity, but the FTBNER dataset does not have it annotated as part of the named-entity, then it will be considered as an incorrect prediction. Take this as an example, considering the first column corresponds to the token, the second column the true NER label and the third column the predicted NER label:

*M. O I-PER*
*Bush I-PER I-PER*

Although the predicted label for "Bush" matches its true label, the whole named-entity needs to match the true label to be considered a correct prediction. Since we cannot change the pre-trained models, another approach to have a more accurate evaluation is to change FTBNER to include the personal titles as part of the named-entities. This was the approach we took, which consisted in finding all personal titles[2] that are followed by a *Person* named-entity and appropriately converting them to be part of the named-entity.

After the pre-processing procedure, the dataset contains $12,351$ sentences with $364,522$ tokens and $11,507$ named-entities. Moreover, there are $2,017$ persons, $3,754$ locations and $5,736$ organizations.

[2] https://fr.wikipedia.org/wiki/Titres_et_pr%C3%A9dicats

Contrary to the training datasets (Section 4.3.4) FTBNER has organizations as the most represented named-entity type.

### 6.1.3 Evaluation Methodology and Metrics

In Section 4.4, we introduced the evaluation methodology and metrics that are going to be applied. The goal is to access the quality of each NER model, when predicting named-entities, by giving each model the sentences present in the FTBNER dataset. The "conlleval" evaluation script will, again, be used to evaluate performance of each model where exact-match evaluation is considered. The metrics used to evaluate each model are *precision*, *recall* and $F1$-*score*, which are presented for each named-entity type and as well as their micro-average (overall score). When selecting the best performing model we aim at maximizing the $F1$-score.

### 6.1.4 Results

The evaluation results for each NER model are represented in Table 6.1. All models show, for locations and persons, a higher recall than precision. This means the models detect more named-entities at the cost of wrongly predicting some. It is also true for locations except for the *flair-pre-trained* model.

The model previously used in CONNECTIONLENS, the *stanford-quaero* model, is, overall, outperformed by all models, having an overall $F1$-score of about $45\%$. It shows very low scores for organizations, with a $F1$-score of $8.04\%$. However, for locations and persons, the scores are comparable to other models' scores. The *flair-ssf-quaero* model, that is described in Chapter 4, also shows low performance for organizations, with a $F1$-score of $30.28\%$. Both the *stanford-quaero* and *flair-ssf-quaero* models are trained on the *Quaero* dataset, which is composed of newspaper issues of the end of the 19th century, where the number of organizations was less than today. That situation combined with the fact that the *Quaero* dataset is smaller than WikiNER, may contribute to the low scores. Moreover, *flair-ssf-quaero* manages to achieve the highest precision and $F1$-score for locations, respectively $68.34\%$ and $71\%$, and recall for persons, $92.67\%$. Despite that, it is the overall second worst model, with an overall $F1$-score of about $55\%$.

The *spacy-pre-trained-sm* model has a slightly better overall performance than *spacy-pre-trained-md*, with an overall $F1$-score difference of $0.28\%$. *spacy-pre-trained-md* shows higher $F1$-scores for locations and organizations, but is worse for people, driving down its overall quality.

The *flair-pre-trained* is the overall best model, not just in $F1$-score, for which it has $71.20\%$, but all overall metrics. Its overall $F1$-score is almost $8\%$ higher than its preceding best model. Moreover, it has the best scores for organizations, with a difference from the other models of more than $10\%$ in $F1$-score and $20\%$ in recall. This model is the best NER model we evaluated.

| flair-pre-trained | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 53.26% | 77.71% | 63.20% |
| ORG | **74.57%** | **75.61%** | **75.09%** |
| PER | **71.76%** | 84.89% | **77.78%** |
| Overall | **65.55%** | **77.92%** | **71.20%** |

| flair-ssf-quaero | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | **68.34%** | 73.87% | **71.00%** |
| ORG | 37.75% | 25.28% | 30.28% |
| PER | 65.91% | **92.67%** | 77.03% |
| Overall | 56.76% | 52.95% | 54.79% |

| spacy-pre-trained-md | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 55.77% | 78.00% | 65.04% |
| ORG | 72.72% | 54.85% | 62.53% |
| PER | 53.09% | 74.98% | 62.16% |
| Overall | 61.06% | 65.93% | 63.40% |

| spacy-pre-trained-sm | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 54.92% | **79.41%** | 64.93% |
| ORG | 71.92% | 53.23% | 61.18% |
| PER | 57.32% | 79.19% | 66.50% |
| Overall | 61.25% | 66.32% | 63.68% |

| stanford-quaero | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 62.17% | 69.05% | 65.43% |
| ORG | 15.82% | 5.39% | 8.04% |
| PER | 55.31% | 88.26% | 68.00% |
| Overall | 50.12% | 40.69% | 44.91% |

**Table 6.1:** NER evaluation results on FTBNER

The pre-trained models are overall better than the models that we trained, i.e., *flair-ssf-quaero* and *stanford-quaero*. Focusing on *flair-ssf-quaero*, it uses the word embedding combination that showed better results in Section 4.5. Considering the results, we decided to train one last model on the WikiNER dataset, in its pre-processed version, using the word embedding combination of *flair-ssf-quaero*, i.e., *stacked forward and backward French Flair embeddings* with *French fastText embeddings*. Moreover, we performed the same steps as in Section 4.7: $(i)$ randomly divided the sentences in the WikiNER dataset in two, $80\%$ for the train set and $20\%$ for the development set, and $(ii)$ set the number of maximum epochs to $150$ while training. The results of evaluating this model, which we call *flair-ssf-wikiner*, on the FTBNER dataset are shown in Table 6.2.

*flair-ssf-wikiner* has an overall $F1$-score of $73.31\%$. We consider *flair-ssf-wikiner* to be even better than *flair-pre-trained*, showing better $F1$-scores, overall and named-entity specific. The most noticeable improvement is in terms of organizations, where the model shows better scores for all metrics, with an improvement of about $6\%$ in precision, $5\%$ in $F1$-score and $2\%$ in recall, when compared to *flair-pre-*

| flair-ssf-wikiner | Precision | Recall | $F_{\beta=1}$ |
|---|---|---|---|
| LOC | 59.52% | 79.36% | 68.02 |
| ORG | **76.56%** | 74.55% | **75.54** |
| PER | **72.29%** | 84.94% | **78.10** |
| Overall | **69.20%** | **77.94%** | **73.31** |

**Table 6.2:** Results of *flair-ssf-wikiner* on FTBNER

*trained*. Moreover, it surpasses *flair-pre-trained* for all scores except the recall of organizations. This is the best performing model evaluated and it was selected to be integrated into CONNECTIONLENS.

## 6.2 Relationship Extraction

This Section explains the experimental evaluation for RE. Section 6.2.1 describes the dataset used. Section 6.2.2 details the methodology and metrics we used for evaluation. Section 6.2.3 presents the training details. Then, Section 6.2.4 gives a description of the dataset variants, i.e., alternative versions, created from the original dataset. Lastly, Section 6.2.5 presents the results obtained with each model variant.

### 6.2.1 Dataset

We train and evaluate our model using the dataset that we created as explained in Chapter 5. The dataset was built using distant supervision by aligning relationships from DBpedia, in its 2016 version, with sentences from Wikipedia articles (also from 2016), in French. There are $30$ relationship types including the $NA$ relationship, that indicates the nonexistence of a relationship between two entities.

The dataset was randomly divided in terms of relationship instances in three sets: $60\%$ of the instances for the train set, $20\%$ of the instances for the development set and $20\%$ of the instances for the test set. The train set contains $59,625$ sentences, $38,064$ entities and $41,536$ relationship instances. The development set contains $14,707$ sentences, $11,451$ entities and $10,269$ relationship instances. And, the test set contains $19,322$ sentences, $14,177$ entities and $13,063$ relationship instances. These values are shown in Table 5.1.

### 6.2.2 Evaluation Methodology and Metrics

Following the evaluation of previous related works, we evaluate our model using *held-out evaluation*. A part of the relationships instances in the KB, in our case DBPedia, is "held-out" to create a test set of relationship instances. Hence, when evaluating the model, the relationships it discovers from test

sentences are compared to the held-out relationship instances. When we divided the model in terms of relationship instances we were already taking into consideration the held-out evaluation.

We evaluate the performance of our model using the following metrics, based on the work of [61] and subsequent works, and based on the capability of the model ranking the relationships it predicts (with a confidence score):

- *Precision-recall (PR) curves* - it shows the precision at different levels of recall. A PR curve [100] can be defined as the set of points for a range of confidence thresholds $c$:

$$PR(\cdot) = \{(recall(c), precision(c)), -\infty < c < +\infty\},$$

  where $recall(c)$ is the portion of positive relationships that are correctly predicted, with a confidence above $c$, and $precision(c)$ is the portion of correctly predicted positive relationships from all relationships predicted as positive, with a confidence above $c$.

- *Area under curve (AUC)* - summarizes the PR curve in a single value. It is also referred to as average precision since it is the weighted average of precisions at each confidence threshold $c$.

- *Micro-$F1$* - there are multiple precision and recalls in this setting. The micro-$F1$ score in this setting is the highest $F1$ score of all precision and recall pairs.

- *Precision@N (P@N)* - is is the precision with the top N highest confidence predictions.

In the test set, there are $10,242$ relationship instances that possess only one sentence, and $2,821$ relationship instances with more than one sentence. Following [61], we evaluate our model on the relationship instances with more than one sentence, to see the effect of the sentence-level *attention* mechanism of our PCNN with selective attention over instances (PCNN-ATT) model, that was built to deal with multiple sentences. Moreover, we evaluate the model on three different test settings: $(i)$ *one*, where for each relationship instance we randomly select one sentence to use for prediction, $(ii)$ *two*, where for each relationship instance we randomly select two sentences to use for prediction, and, $(iii)$ *all*, where all sentences of the relationship instances are used for prediction. We compute P@100, P@200, P@300, and their means, using the three test sets that result from the different settings described. We also compute, for each test set setting, the AUC and micro-$F1$.

### 6.2.3 Training Details

We use OpenNRE [6] to train a bag-level PCNN-ATT model, the same architecture as the model proposed in [61], as we have mentioned in Section 5.6. For training, we use a maximum of $30$ epochs, using the train set. Each epoch generates a model and, in the end, the model that gives the best AUC on the

development set is kept. In what concerns the remaining hyperparameters, we use OpenNRE's default hyperparameters for bag-level training.

To train the final model we use the complete dataset and a maximum of $150$ epochs. However, it is still necessary to keep a portion of the dataset to be used as the development set. Therefore, we randomly divide the dataset, again, in train and development sets, keeping $80\%$ of the sentences for the train set and $20\%$ of the sentences for the development set.

The use of a bag-level model restricts our word embedding use to traditional, static word embeddings. Dynamic word embeddings, e.g., BERT, or specifically for French, CamemBERT, are not possible to use in combination with bag-level training due to it being too hard and due to memory limits. OpenNRE bag-level training only allows using static word embeddings for this reason, and, although the developers of OpenNRE managed to train a bag-level BERT limited to a bag size of less than $3$, it underperformed compared to not using BERT. In conclusion, we used the freely available fastText word embeddings in French, trained on Wikipedia[3].

### 6.2.4 Experimenting with Dataset Variants

In Section 5.3 we grouped similar DBpedia relationships which coarsely have the same meaning. The *locatedIn* relationship is the relationship with the highest amount of relationship instances and sentences. It practically includes all original DBpedia relationships that mean that a location or an organization is part of/located in a location. We grouped those relationships because we consider them to have a similar meaning, however, there is a possibility that there is some underlying differences between certain types of relationships that were grouped in the *locatedIn* relationship and the model might be able to make more accurate predictions if they are separate. For this reason, we decided to experiment with alternative versions of the dataset.

We call the original dataset we have previously described by *version 1* (v1). For *version 2* (v2) we separated the *locatedIn* relationship in two different relationships: *partOf* for relationships where both entities are locations and *locatedIn* when $ent_1$ is an organization and $ent_2$ is a location. Originally, there are $22,051$ relationship instances and $26,447$ sentences for the *locatedIn* relationship type. In "version 2", there are $19,901$ relationship instances and $23,588$ sentences for the relationship *partOf* and $2,150$ relationship instances and $2,859$ sentences for the relationship *locatedIn*.

For *version 3* (v3) we took the original dataset, and separated the *capital* relationship from where it was inserted into, i.e., *locatedIn*, to be a separate relationship. Moreover, there are $58$ relationship instances and $157$ sentences for the relationship *capital*.

The goal of these dataset variants is to examine if the changes in the dataset significantly affect the performance of the model. We train three different models and the results obtained are presented and

[3] https://fasttext.cc/docs/en/pretrained-vectors.html

analyzed in Section 6.2.5.

### 6.2.5 Results

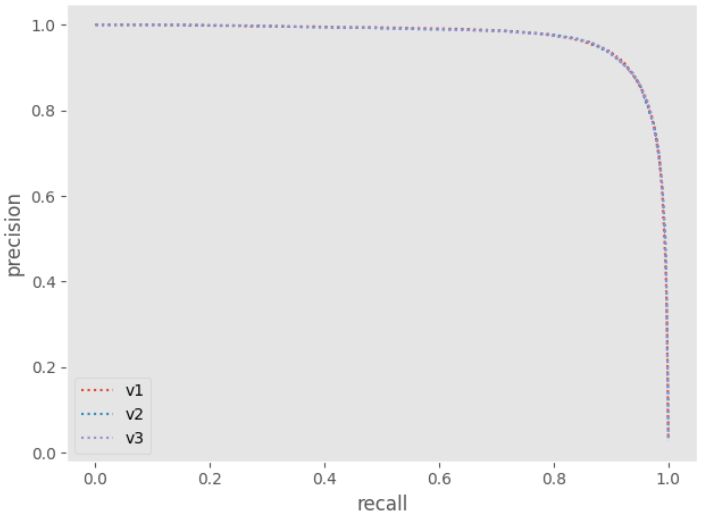


**Figure 6.1:** PR curves of RE models

| | AUC | micro-$F1$ |
|---|---|---|
| v1 | **97.10%** | **91.78%** |
| v2 | 97.09% | 91.61% |
| v3 | 97.06% | 91.65% |

**Table 6.3:** AUC and micro-$F1$ of RE models

In Figure 6.1 the PR curves of the different models we trained are shown. We can see how each model variant performs over different recall levels. There is hardly no difference between the models performance. This means that the changes made to the original dataset, overall, have no effect on the precision and recall. Moreover, Table 6.3 shows a comparison of the AUC and micro-$F1$ values for the model variants. The original version, *v1*, is slightly better that the other versions, *v2* and *v3*, in terms of AUC and micro-$F1$, with $97.10\%$ and $91.78\%$, respectively. The high values of AUC all models share mean both a high recall and a high precision. Again, the variants applied to the dataset do not make, overall, a significant difference. The good results produced by the models are influenced by the fact that the train set and the test set are chunks of the same dataset. We checked, and there is no test leak problem, i.e., there are no relationship instances in the test set that were used for training.

In Table 6.4 the P@100, P@200, P@300, AUC and micro-$F1$ with one, two and all sentences for each entity pair, for all model variants are presented. For all models, increasing the number of sentences improves the results, particularly in terms of AUC and micro-$F1$. From the *one* test setting to the *all* test

| | One | | | | | |
|---|---|---|---|---|---|---|
| | P@100 | P@200 | P@300 | P@Mean | AUC | micro-$F1$ |
| v1 | **99%** | **99.5%** | **99%** | **99.2%** | 88.64% | 81.18% |
| v2 | **99%** | 98% | 97.7% | 98.2% | 88.25% | 80.87% |
| v3 | **99%** | **99.5%** | 98.7% | 99.1% | **88.96%** | **81.86%** |
| | Two | | | | | |
| | P@100 | P@200 | P@300 | P@Mean | AUC | micro-$F1$ |
| v1 | 99% | 99.5% | 99.3% | 99.3% | 92.05% | 84.36% |
| v2 | **100%** | **100%** | **99.7%** | **99.9%** | 92.18% | **85.18%** |
| v3 | 99% | 99.5% | 99% | 99.2% | **92.37%** | 85.01% |
| | All | | | | | |
| | P@100 | P@200 | P@300 | P@Mean | AUC | micro-$F1$ |
| v1 | 99% | **99.5%** | 99.3% | 99.3% | **93.28%** | 85.99% |
| v2 | **100%** | **99.5%** | **99.7%** | **99.7%** | 93.16% | **86.08%** |
| v3 | 99% | **99.5%** | 99.3% | 99.3% | 93.03% | 85.96% |

**Table 6.4:** P@N, AUC and micro-$F1$ for different number of sentences in bags

setting there is an improvement of around $5\%$ for micro-$F1$, for all models. Moreover, improvements are noticeable just by increasing the number of sentences from one to two.

In conclusion, we decided to select the *v1* model, i.e., the model trained on the original dataset, although, the differences in performance between all models are not significant. Moreover, *v1* has the best overall AUC and micro-$F1$. When evaluating the models on different number of sentences in a bag: $(i)$ it has the best P@N for all N values in the *one* test setting, $(ii)$ it has the best AUC when using all sentences, i.e., in the *all* test setting. With *v1* we also do not have the *locatedIn* relationship divided in other relationships, which may confuse the model if the relationships prove to be too similar.

## 6.3 Integration in ConnectionLens

The goal of the thesis is to develop a solution for NER and RE for French news texts that can be integrated in CONNECTIONLENS. The solution consisted in developing machine learning models for each task. Both the NER and the RE models we developed and selected require Python packages. However, CONNECTIONLENS is implemented in Java. To integrate the selected NER and RE models, we create a micro web-service, running with the CONNECTIONLENS code, using Flask[4] [101].

The Flask service starts by loading the models and waits for HTTP requests, that are issued by CONNECTIONLENS. When CONNECTIONLENS needs to extract named-entities and/or relationships from text an HTTP POST request is sent to the service.

In particular for NER, an HTTP POST request contains the input text, which the service receives.

[4]https://flask.palletsprojects.com/

The service starts by tokenizing and segmenting the text into sentences, then, provides the tokenized sentences to the model to make predictions. These predictions are then sent to CONNECTIONLENS in an HTTP response in JSON, that is composed of a JSON object containing an array of sentences and their respective predictions. Each prediction, in the 'entities' array, contains the named-entity itself, its position in the sentence and the predicted label with an associated confidence score. Below is an example of the returned JSON for the sentence "Barack Hussein Obama II, né le 4 août 1961 à Honolulu.":

```
{
"sentences":[
   {
   "text":"Barack Hussein Obama II , né le 4 août 1961 à Honolulu .",
   "labels":[],
   "entities":[
       { "text":"Barack Hussein Obama II",
       "start_pos":0,
       "end_pos":23,
       "type":"PER",
       "confidence":0.7960530519485474 },
       { "text":"Honolulu",
       "start_pos":46,
       "end_pos":54,
       "type":"LOC",
       "confidence":0.9999279975891113 } ]
   }
] }
```

In what concerns RE, the integration with CONNECTIONLENS is not accomplished. The idea is to still use the Flask service for the integration and make use of CONNECTIONLENS capabilities of node matching, that link nodes whose data is considered to be similar, as we exemplified in Chapter 1, to group all sentences that contain the same two entities, i.e., a bag. The service will load the RE model and wait for an HTTP POST request to be made by CONNECTIONLENS with a bag of sentences with the following structure:

```
[{
   'text':  ...,
   'h':  {'pos':  [start, end]},
   't':  {'pos':  [start, end]}
}
...
```

]

The array corresponds to the bag of sentences for the same two entities. Each object in the array corresponds to a different sentence where the entities co-occur. Moreover, 'text' corresponds to the sentence itself, 'h' corresponds to the head entity, i.e., $ent_1$, 't' corresponds to the tail entity, i.e., $ent_2$, and where 'pos' corresponds the position of the entity in the sentence, character-wise.

The predicted relationship, for the entity pair, and its corresponding confidence score will be returned to CONNECTIONLENS in an HTTP response.

# 7

# Conclusion

## Contents

In this thesis, we propose the development of a solution for NER and RE for French news text, to be integrated in CONNECTIONLENS. This entails discovering the best approach for performing NER and RE, that can be applied effectively and efficiently to French texts. This chapter presents the main conclusions and suggestions for future work.

## 7.1 Summary

For both NER and RE, implementing a technique from scratch was not deemed necessary due to the availability of tools, namely third-party libraries that allow one to train a model by just providing the, usually labeled, data.

As described in Chapter 3, in what concerns NER, we found black-box tools and third-party libraries with pre-trained models that can perform French NER off-the-shelf. Black-box tools are usually a web service accessible by an API, and impose constraints like a maximum number of accesses per a certain time period. The constraints are not desired for CONNECTIONLENS, so we turned to third-party libraries. Both Flair and SpaCy use neural-based state-of-the-art techniques and have French NER pre-trained models available.

We decided to create our own French NER model using the best model we could create using Flair and SpaCy, to see if we could improve upon their pre-trained models. We describe the creation of this model in Chapter 4. We pre-processed several datasets with the goal of combining them for training our models, but in the end we ended up training the models using a pre-processed version of the *Quaero Old Press Extended Named Entity* dataset. After experimenting with different configurations, e.g., word embeddings, for training models using Flair and SpaCy, we concluded and selected as the best performing model, a Flair model trained using the Quaero dataset and stacked forward and backward French Flair embeddings with French fastText embeddings, which we named *flair-ssf-quaero*. It achieved an overall $F1$-score of $82.44\%$ on the test set of the pre-processed Quaero dataset.

We evaluated this model along with the pre-trained models, and the model previously present in CONNECTIONLENS on the FTBNER dataset, whose results are presented in Chapter 6. Considering the results we decided to train one last model on the WikiNER dataset using the word embedding combination of *flair-ssf-quaero*, which we named *flair-ssf-wikiner*. This resulted in *flair-ssf-wikiner* being the best model evaluated, with an overall $F1$-score of $73.31\%$ on the FTBNER dataset, and was selected to integrate CONNECTIONLENS.

Regarding RE, the tools we found are described in Chapter 3. We found two tools capable of dealing with French, off-the-shelf: IBM Watson NLU and the French version of ReVerb. IBM Watson NLU is a web service and thus impose constraints not desirable for CONNECTIONLENS. On the other hand, the French version of ReVerb is an Open IE system and the fact that there is not a defined set of

relationships that can be extracted is, likewise, not desired for CONNECTIONLENS. Moreover, we found third-party libraries that, although only having pre-trained models available for English, allow one to train their own model. Since datasets labeled with entities and relationships do not exist for French, we are limited to using a tool that implements a technique that does not require manually labeled data. The only tool we found was OpenNRE, that besides allowing training supervised models, also allows training distantly supervised models, by implementing bag-level training.

Therefore, we decided to train a bag-level model using OpenNRE with a dataset we built using distant supervision, that we describe in Chapter 5. We used French DBpedia to obtain relationship instances and French Wikipedia articles to extract sentences expressing those relationships. We filtered and grouped the DBpedia relationships and that result in $30$ relationships types including the $NA$ relationship, that indicates the nonexistence of a relationship between two entities.

We perform an evaluation of our RE extraction solution in Chapter 6. Moreover, after building the dataset we experimented with alternative versions of the dataset where small changes were made to the relationship groups. Each variant of the dataset resulted in a different model which we evaluated. The best performing model was the one trained on the original version of the dataset, which achieved an AUC and a micro-$F1$ of $97.10\%$ and $91.78\%$, respectively, and that we selected to integrate CONNECTIONLENS.

In what concerns RE, the integration with CONNECTIONLENS is not accomplished. On the other hand, the NER solution is integrated and included in the paper [8], accepted to the 36th Conference "Gestion de Données - Principes, Technologies et Applications" (BDA 2020), where the NER model and experiments are described along with the approach of CONNECTIONLENS.

## 7.2 Future Work

In terms of future work, the integration of the RE solution in CONNECTIONLENS as described in Section 6.3, is going to be accomplished in the near future.

Possible future work for NER could be:

- attempting to improve the results by using other encoding schemes for the training datasets.
- combining the WikiNER dataset with the Quaero dataset to train a model, to see the effect of using more data as it was originally planned in Chapter 4.
- using a partial-match evaluation instead of exact-match, so the evaluation is not so strict and allows partial matches, where, for example the boundaries of the named-entity may not be entirely correct according to the true annotation of the dataset.

Possible future work for RE could be:

- training the model with more relationship instances and sentences.
- using a text corpus other than Wikipedia to create the dataset, a harder and more real scenario. Or, creating a test set based on a distinct text corpus, from another domain, e.g., news.
- performing manual evaluation of the RE models.
- presenting some of the evaluation metrics, namely, AUC and micro-$F1$, per relationship type.

Another experiment we would also like to devise for RE consists in the use of French contextual word embeddings, e.g., CamemBERT, and not just static, traditional word embeddings. This would, currently, only be possible in a supervised sentence-level approach. Moreover, we already prepared the code to use CamemBERT in OpenNRE in a sentence-level setting. In addition we would create another dataset, using DBPedia and Wikipedia, and the distant supervision procedure, however, instead of distant supervision assumptions, we would make the assumption that *if two entities are related in the KB, there is only one sentence that expresses the relationship*, and select only the first sentence where the entities co-occur, inspired by [92]. Due to Wikipedia's standardized nature and the particularity that its sentences usually express facts, we can assume that the first sentence where the given two entities co-occur, most probably is the only one that expresses the relationship.

In general, for both the NER and RE models, we would like to experiment more in regard to hyper-parameter optimization.

# Bibliography


[1] J. Gray, L. Chambers, and L. Bounegru, *The Data Journalism Handbook: How Journalists Can Use Data to Improve the News*. O'Reilly Media, 2012.

[2] C. Chanial, R. Dziri, H. Galhardas, J. Leblay, M.-H. L. Nguyen, and I. Manolescu, "Connectionlens: Finding connections across heterogeneous data sources," *Proc. VLDB Endow.*, vol. 11, pp. 2030–2033, Aug. 2018.

[3] S. Sarawagi, "Information extraction," *Foundations and Trends in Databases*, vol. 1, no. 3, pp. 261–377, 2008.

[4] A. Akbik, T. Bergmann, D. Blythe, K. Rasul, S. Schweter, and R. Vollgraf, "FLAIR: An easy-to-use framework for state-of-the-art NLP," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, (Minneapolis, Minnesota), pp. 54–59, Association for Computational Linguistics, June 2019.

[5] F. Gotti and P. Langlais, "Harnessing open information extraction for entity classification in a french corpus," in *Canadian AI 2016*, Springer International Publishing Switzerland, Springer International Publishing Switzerland, 2016.

[6] X. Han, T. Gao, Y. Yao, D. Ye, Z. Liu, and M. Sun, "OpenNRE: An open and extensible toolkit for neural relation extraction," in *Proceedings of EMNLP-IJCNLP: System Demonstrations*, pp. 169–174, 2019.

[7] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, and C. Bizer, "Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web Journal*, vol. 6, 01 2014.

[8] O. Balalau, C. Conceição, H. Galhardas, I. Manolescu, T. Merabti, J. You, and Y. Youssef, "Graph integration of structured, semistructured and unstructured data for data journalism," *BDA 2020 (informal publication)*.

[9] M.-F. Moens, *Information Extraction: Algorithms and Prospects in a Retrieval Context*, vol. 21. 01 2006.

[10] A. McCallum, "Information extraction: Distilling structured data from unstructured text," *Queue*, vol. 3, pp. 4:48–4:57, Nov. 2005.

[11] D. Jurafsky and J. H. Martin, *Speech and Language Processing (3rd Edition, draft)*. 2018.

[12] D. Campos, S. Matos, and J. L. Oliveira, "Biomedical named entity recognition: A survey of machine-learning tools," in *Theory and Applications for Advanced Text Mining* (S. Sakurai, ed.), ch. 8, Rijeka: IntechOpen, 2012.

[13] J. Pustejovsky and A. Stubbs, *Natural Language Annotation for Machine Learning*. No. v. 9, p. 878 in Natural Language Annotation for Machine Learning, O'Reilly Media, Incorporated, 2012.

[14] L. Ramshaw and M. Marcus, "Text chunking using transformation-based learning," in *Third Workshop on Very Large Corpora*, 1995.

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

[16] D. Nadeau and S. Sekine, "A survey of named entity recognition and classification," *Lingvisticae Investigationes*, vol. 30, no. 1, pp. 3–26, 2007.

[17] R. Sharnagat, "Named entity recognition: A literature survey," 2014.

[18] J. Patrick, C. Whitelaw, and R. Munro, "Slinerc: The sydney language-independent named entity recogniser and classifier," in *proceedings of the 6th conference on Natural language learning-Volume 20*, pp. 1–4, Association for Computational Linguistics, 2002.

[19] Y. Goldberg, *Neural Network Methods for Natural Language Processing*, vol. 37 of *Synthesis Lectures on Human Language Technologies*. San Rafael, CA: Morgan & Claypool, 2017.

[20] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state markov chains," *The annals of mathematical statistics*, vol. 37, no. 6, pp. 1554–1563, 1966.

[21] L. E. Baum and J. A. Eagon, "An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology," *Bulletin of the American Mathematical Society*, vol. 73, no. 3, pp. 360–363, 1967.

[22] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, pp. 257–286, Feb 1989.

[23] G. D. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[24] A. Ratnaparkhi, “A maximum entropy model for part-of-speech tagging,” in *Conference on Empirical Methods in Natural Language Processing*, 1996.

[25] A. McCallum, D. Freitag, and F. C. Pereira, “Maximum entropy markov models for information extraction and segmentation.,”

[26] J. Lafferty, A. McCallum, and F. C. Pereira, “Conditional random fields: Probabilistic models for segmenting and labeling sequence data,” 2001.

[27] C. Sutton and A. Mccallum, *An Introduction to Conditional Random Fields for Relational Learning*. 01 2007.

[28] C. Sutton, A. McCallum, *et al.*, “An introduction to conditional random fields,” *Foundations and Trends® in Machine Learning*, vol. 4, no. 4, pp. 267–373, 2012.

[29] H. M. Wallach, “Conditional random fields: An introduction,” *Technical Reports (CIS)*, p. 22, 2004.

[30] J. Li, A. Sun, J. Han, and C. Li, “A survey on deep learning for named entity recognition,” *arXiv preprint arXiv:1812.09449*, 2018.

[31] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.

[32] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, “Neural architectures for named entity recognition,” *arXiv preprint arXiv:1603.01360*, 2016.

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

[34] J. Pennington, R. Socher, and C. Manning, “Glove: Global vectors for word representation,” in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

[35] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, “Enriching word vectors with subword information,” *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017.

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

[37] A. Akbik, D. Blythe, and R. Vollgraf, “Contextual string embeddings for sequence labeling,” in *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 1638–1649, 2018.

[38] R. Al-Rfou, V. Kulkarni, B. Perozzi, and S. Skiena, “Polyglot-ner: Massive multilingual named entity recognition,” in *Proceedings of the 2015 SIAM International Conference on Data Mining*, pp. 586–594, SIAM, 2015.

[39] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, “The stanford corenlp natural language processing toolkit,” in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pp. 55–60, 2014.

[40] J. R. Finkel, T. Grenager, and C. Manning, “Incorporating non-local information into information extraction systems by gibbs sampling,” in *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, ACL ’05, (Stroudsburg, PA, USA), pp. 363–370, Association for Computational Linguistics, 2005.

[41] J. Nothman, N. Ringland, W. Radford, T. Murphy, and J. R. Curran, “Learning multilingual named entity recognition from wikipedia,” *Artificial Intelligence*, vol. 194, pp. 151–175, 2013.

[42] F. Dernoncourt, J. Y. Lee, and P. Szolovits, “NeuroNER: an easy-to-use program for named-entity recognition based on neural networks,” *Conference on Empirical Methods on Natural Language Processing (EMNLP)*, 2017.

[43] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[44] S. Brin, “Extracting patterns and relations from the world wide web,” in *The World Wide Web and Databases* (P. Atzeni, A. Mendelzon, and G. Mecca, eds.), (Berlin, Heidelberg), pp. 172–183, Springer Berlin Heidelberg, 1999.

[45] E. Agichtein and L. Gravano, “Snowball: Extracting relations from large plain-text collections,” in *Proceedings of the fifth ACM conference on Digital libraries*, pp. 85–94, ACM, 2000.

[46] S. Pawar, G. K. Palshikar, and P. Bhattacharyya, “Relation extraction: A survey,” *arXiv preprint arXiv:1712.05191*, 2017.

[47] D. Ravichandran and E. Hovy, “Learning surface text patterns for a question answering system,” in *Proceedings of the 40th Annual meeting of the association for Computational Linguistics*, pp. 41–47, 2002.

[48] P. Pantel and M. Pennacchiotti, "Espresso: Leveraging generic patterns for automatically harvesting semantic relations," in *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pp. 113–120, 2006.

[49] R. Gabbard, M. Freedman, and R. Weischedel, "Coreference for learning to extract relations: Yes virginia, coreference matters," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 288–293, 2011.

[50] A. Sun, R. Grishman, and S. Sekine, "Semi-supervised relation extraction with large-scale word clustering," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, (Portland, Oregon, USA), pp. 521–529, Association for Computational Linguistics, June 2011.

[51] M. Mintz, S. Bills, R. Snow, and D. Jurafsky, "Distant supervision for relation extraction without labeled data," in *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pp. 1003–1011, Association for Computational Linguistics, 2009.

[52] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 1247–1250, 2008.

[53] D. Vrandečić and M. Krötzsch, "Wikidata: a free collaborative knowledgebase," *Communications of the ACM*, vol. 57, no. 10, pp. 78–85, 2014.

[54] F. M. Suchanek, G. Kasneci, and G. Weikum, "Yago: a core of semantic knowledge," in *Proceedings of the 16th international conference on World Wide Web*, pp. 697–706, 2007.

[55] R. Hoffmann, C. Zhang, and D. S. Weld, "Learning 5000 relational extractors," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, (Uppsala, Sweden), pp. 286–295, Association for Computational Linguistics, July 2010.

[56] S. Riedel, L. Yao, and A. McCallum, "Modeling relations and their mentions without labeled text," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 148–163, Springer, 2010.

[57] R. Hoffmann, C. Zhang, X. Ling, L. Zettlemoyer, and D. S. Weld, "Knowledge-based weak supervision for information extraction of overlapping relations," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, (Portland, Oregon, USA), pp. 541–550, Association for Computational Linguistics, June 2011.

[58] M. Surdeanu, J. Tibshirani, R. Nallapati, and C. D. Manning, “Multi-instance multi-label learning for relation extraction,” in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, (Jeju Island, Korea), pp. 455–465, Association for Computational Linguistics, July 2012.

[59] D. Zeng, K. Liu, Y. Chen, and J. Zhao, “Distant supervision for relation extraction via piecewise convolutional neural networks,” in *Proceedings of the 2015 conference on empirical methods in natural language processing*, pp. 1753–1762, 2015.

[60] D. Zeng, K. Liu, S. Lai, G. Zhou, J. Zhao, *et al.*, “Relation classification via convolutional deep neural network,” 2014.

[61] Y. Lin, S. Shen, Z. Liu, H. Luan, and M. Sun, “Neural relation extraction with selective attention over instances,” in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (Berlin, Germany), pp. 2124–2133, Association for Computational Linguistics, Aug. 2016.

[62] S. Vashishth, R. Joshi, S. S. Prayaga, C. Bhattacharyya, and P. Talukdar, “RESIDE: Improving distantly-supervised neural relation extraction using side information,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, (Brussels, Belgium), pp. 1257–1266, Association for Computational Linguistics, Oct.-Nov. 2018.

[63] Z.-X. Ye and Z.-H. Ling, “Distant supervision relation extraction with intra-bag and inter-bag attentions,” *arXiv preprint arXiv:1904.00143*, 2019.

[64] P. Xu and D. Barbosa, “Connecting language and knowledge with heterogeneous representations for neural relation extraction,” *arXiv preprint arXiv:1903.10126*, 2019.

[65] M. Banko, M. J. Cafarella, S. Soderland, M. Broadhead, and O. Etzioni, “Open information extraction from the web.,”

[66] C. Niklaus, M. Cetto, A. Freitas, and S. Handschuh, “A survey on open information extraction,” in *Proceedings of the 27th International Conference on Computational Linguistics*, (Santa Fe, New Mexico, USA), pp. 3866–3878, Association for Computational Linguistics, Aug. 2018.

[67] M. Banko and O. Etzioni, “The tradeoffs between open and traditional relation extraction,” in *Proceedings of ACL-08: HLT*, (Columbus, Ohio), pp. 28–36, Association for Computational Linguistics, June 2008.

[68] F. Wu and D. S. Weld, “Open information extraction using Wikipedia,” in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, (Uppsala, Sweden), pp. 118–127, Association for Computational Linguistics, July 2010.

[69] A. Fader, S. Soderland, and O. Etzioni, “Identifying relations for open information extraction,” in *Proceedings of the conference on empirical methods in natural language processing*, pp. 1535–1545, Association for Computational Linguistics, 2011.

[70] M. Schmitz, R. Bart, S. Soderland, O. Etzioni, *et al.*, “Open language learning for information extraction,” in *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 523–534, Association for Computational Linguistics, 2012.

[71] M. Yahya, S. Whang, R. Gupta, and A. Halevy, “Renoun: Fact extraction for nominal attributes,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 325–335, 2014.

[72] A. Akbik and A. Löser, “Kraken: N-ary facts in open information extraction,” in *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction (AKBC-WEKEX)*, pp. 52–56, 2012.

[73] F. Mesquita, J. Schmidek, and D. Barbosa, “Effectiveness and efficiency of open relation extraction,” in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 447–457, 2013.

[74] M. Mausam, “Open information extraction systems and downstream applications,” in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI’16, p. 4074–4077, AAAI Press, 2016.

[75] J. Christensen, Mausam, S. Soderland, and O. Etzioni, “Semantic role labeling for open information extraction,” in *Proceedings of the NAACL HLT 2010 First International Workshop on Formalisms and Methodology for Learning by Reading*, (Los Angeles, California), pp. 52–60, Association for Computational Linguistics, June 2010.

[76] H. Pal *et al.*, “Demonyms and compound relational nouns in nominal open ie,” in *Proceedings of the 5th workshop on automated knowledge base construction*, pp. 35–39, 2016.

[77] H. Bast and E. Haussmann, “Open information extraction via contextual sentence decomposition,” in *2013 IEEE Seventh International Conference on Semantic Computing*, pp. 154–159, IEEE, 2013.

[78] N. Bhutani, H. V. Jagadish, and D. Radev, “Nested propositions in open information extraction,” in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, (Austin, Texas), pp. 55–64, Association for Computational Linguistics, Nov. 2016.

[79] L. Del Corro and R. Gemulla, “Clausie: clause-based open information extraction,” in *Proceedings of the 22nd international conference on World Wide Web*, pp. 355–366, 2013.

[80] G. Angeli, M. J. J. Premkumar, and C. D. Manning, “Leveraging linguistic structure for open domain information extraction,” in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, vol. 1, pp. 344–354, 2015.

[81] K. Gashteovski, R. Gemulla, and L. d. Corro, “Minie: minimizing facts in open information extraction,” Association for Computational Linguistics, 2017.

[82] M. Cetto, C. Niklaus, A. Freitas, and S. Handschuh, “Graphene: Semantically-linked propositions in open information extraction,” *arXiv preprint arXiv:1807.11276*, 2018.

[83] G. Stanovsky, J. Michael, L. Zettlemoyer, and I. Dagan, “Supervised open information extraction,” in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 885–895, 2018.

[84] J. Michael, G. Stanovsky, L. He, I. Dagan, and L. Zettlemoyer, “Crowdsourcing question-answer meaning representations,” *arXiv preprint arXiv:1711.05885*, 2017.

[85] L. Cui, F. Wei, and M. Zhou, “Neural open information extraction,” *arXiv preprint arXiv:1805.04270*, 2018.

[86] M. Surdeanu, D. McClosky, M. R. Smith, A. Gusev, and C. D. Manning, “Customizing an information extraction system to a new domain,” in *Proceedings of the ACL 2011 Workshop on Relational Models of Semantics*, pp. 2–10, Association for Computational Linguistics, 2011.

[87] T. Kiss and J. Strunk, “Unsupervised multilingual sentence boundary detection,” *Computational linguistics*, vol. 32, no. 4, pp. 485–525, 2006.

[88] M. Marcus, B. Santorini, and M. A. Marcinkiewicz, “Building a large annotated corpus of english: The penn treebank,” 1993.

[89] C. Neudecker, “An open corpus for named entity recognition in historic newspapers,” in *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC’16)*, (Portorož, Slovenia), pp. 4348–4352, European Language Resources Association (ELRA), May 2016.

[90] O. Galibert, S. Rosset, C. Grouin, P. Zweigenbaum, and L. Quintard, “Extended named entity annotation on ocred documents: From corpus constitution to evaluation campaign,”

[91] L. Ratinov and D. Roth, “Design challenges and misconceptions in named entity recognition,” in *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, (Boulder, Colorado), pp. 147–155, Association for Computational Linguistics, June 2009.

[92] A. P. Aprosio, C. Giuliano, and A. Lavelli, “Extending the coverage of dbpedia properties using distant supervision over wikipedia.,”

[93] T. Nunes and D. Schwabe, “Building distant supervised relation extractors,” in *2014 IEEE International Conference on Semantic Computing*, pp. 44–51, IEEE, 2014.

[94] M. Zając and A. Przepiórkowski, “Distant supervision learning of dbpedia relations,” in *International Conference on Text, Speech and Dialogue*, pp. 193–200, Springer, 2013.

[95] M. Dojchinovski, J. Hernandez, M. Ackermann, A. Kirschenbaum, and S. Hellmann, “Dbpedia nif: Open, large-scale and multilingual knowledge extraction corpus,” *arXiv preprint arXiv:1812.10315*, 2018.

[96] B. Sagot, M. Richard, and R. Stern, “Annotation référentielle du corpus arboré de paris 7 en entités nommées (referential named entity annotation of the paris 7 French TreeBank) [in French],” in *Proceedings of the Joint Conference JEP-TALN-RECITAL 2012, volume 2: TALN*, (Grenoble, France), pp. 535–542, ATALA/AFCP, June 2012.

[97] M. Candito and B. Crabbé, “Improving generative statistical parsing with semi-supervised word clustering,” 2009.

[98] A. Abeillé, L. Clément, and F. Toussenel, “Building a treebank for french,” in *Treebanks*, pp. 165–187, Springer, 2003.

[99] L. Martin, B. Muller, P. J. Ortiz Suárez, Y. Dupont, L. Romary, É. de la Clergerie, D. Seddah, and B. Sagot, “CamemBERT: a tasty French language model,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, (Online), pp. 7203–7219, Association for Computational Linguistics, July 2020.

[100] K. Boyd, K. H. Eng, and C. D. Page, “Area under the precision-recall curve: point estimates and confidence intervals,” in *Joint European conference on machine learning and knowledge discovery in databases*, pp. 451–466, Springer, 2013.

[101] M. Grinberg, *Flask web development: developing web applications with python*. " O’Reilly Media, Inc.", 2018.

# A

# RE Dataset statistics

## A.1 # entities per type and set

| entity type | train | test | dev |
|---|---|---|---|
| PER | 17,182 | 6,083 | 4,833 |
| LOC | 15,598 | 5,818 | 4,827 |
| ORG | 5,284 | 2,276 | 1,791 |
| total | 38,064 | 14,177 | 11,451 |

## A.2 # relationship instances per type and set

| relation | train | test | dev |
|---|---|---|---|
| locatedIn | 14,063 | 4,432 | 3,556 |
| birthPlace | 9,619 | 2,994 | 2,362 |
| NA | 5,874 | 1,849 | 1,447 |

Continued on next page

| relation | train | test | dev |
|---|---|---|---|
| deathPlace | 2,943 | 931 | 711 |
| party | 1,652 | 544 | 416 |
| familyMember | 1,053 | 334 | 292 |
| parentOrganisation | 907 | 286 | 236 |
| bandMember | 766 | 254 | 173 |
| sportsTeam | 676 | 202 | 152 |
| studiedAt | 624 | 162 | 130 |
| childOrganisation | 564 | 187 | 152 |
| nearTo | 525 | 172 | 137 |
| owner | 298 | 110 | 58 |
| spouse | 278 | 81 | 67 |
| influencedBy | 218 | 64 | 56 |
| leader | 205 | 67 | 50 |
| affiliatedWith | 162 | 54 | 38 |
| mentorOf | 148 | 47 | 28 |
| residesIn | 139 | 52 | 36 |
| employedBy | 126 | 43 | 24 |
| foundedBy | 124 | 36 | 27 |
| mentoredBy | 123 | 35 | 28 |
| recordLabel | 109 | 28 | 24 |
| influenced | 95 | 28 | 19 |
| coach | 74 | 23 | 15 |
| architect | 53 | 13 | 14 |
| operator | 41 | 11 | 8 |
| keyPerson | 41 | 14 | 7 |
| created | 25 | 6 | 4 |
| knownFor | 11 | 4 | 2 |
| total | 41,536 | 13,063 | 10,269 |

## A.3 # sentences per type and set

| relationship type | train | test | dev |
|---|---|---|---|
| locatedIn | 16,774 | 5,337 | 4,336 |
| birthPlace | 11,956 | 3,798 | 2,970 |
| NA | 8,551 | 2,792 | 2,120 |
| deathPlace | 4,034 | 125 | 1,000 |
| familyMember | 2,968 | 990 | 756 |
| party | 2,840 | 1,012 | 710 |
| bandMember | 2,059 | 705 | 354 |
| parentOrganisation | 2,003 | 656 | 504 |
| childOrganisation | 1,304 | 514 | 402 |
| nearTo | 1,039 | 348 | 256 |
| sportsTeam | 933 | 250 | 205 |
| spouse | 752 | 263 | 196 |
| studiedAt | 718 | 187 | 139 |
| owner | 711 | 223 | 95 |

Continued on next page

| relationship type | train | test | dev |
|---|---|---|---|
| leader | 466 | 160 | 125 |
| influencedBy | 365 | 115 | 98 |
| affiliatedWith | 269 | 75 | 53 |
| employedBy | 261 | 60 | 40 |
| foundedBy | 249 | 75 | 49 |
| mentorOf | 233 | 106 | 46 |
| residesIn | 203 | 78 | 59 |
| influenced | 198 | 58 | 50 |
| mentoredBy | 188 | 99 | 37 |
| recordLabel | 134 | 32 | 32 |
| coach | 134 | 52 | 20 |
| keyPerson | 99 | 41 | 14 |
| architect | 72 | 19 | 21 |
| operator | 59 | 12 | 5 |
| created | 35 | 7 | 13 |
| knownFor | 18 | 8 | 2 |
| total | 59,625 | 19,322 | 14,707 |