IAM at IberLEF 2022: NER of Species Mentions

Sébastien Cossin[1], Gayo Diallo[1,2] and Vianney Jouhet[1,2]

[1]Univ. Bordeaux, Inserm, Bordeaux Population Health Research Center, team AHEAD, UMR 1219, F-33000 Bordeaux, France

[2]CHU de Bordeaux, Pôle de santé publique, Service d'information médicale, Informatique et Archivistique Médicales (IAM), F-33000 Bordeaux, France

Abstract

In this paper, we describe the approach and the results of our participation in task 1: LivingNER-Species NER track (Species mention entity recognition) of the LivingNER shared task. We tackled the task of automatically detecting species mention in Spanish clinical case reports. We used a dictionary-based approach using only the materials provided by the task organizers. The training set consisted of 1,500 clinical cases annotated by clinical experts. Our system achieved an F1-score of 0.8965 on a test set of 500 clinical cases.

Keywords

named entity recognition, natural language processing, animals, medical informatics

1. Introduction

Detection and identification of species or living organisms in documents is critical in public health surveillance systems that track the emergence of infectious diseases[1, 2]. Most medical information is provided by healthcare professionals in the form of free text, which has many advantages such as familiarity, ease of use and freedom to express complex things, but free text can be very difficult for algorithms to understand[3]. Natural language processing (NLP) develops methods for managing free-text data and extracting information required by applications such as public health surveillance systems. A frequent step in a NLP pipeline is the detection of medical entities (treatment, diagnosis) with named entity recognition (NER) algorithms[4]. LivingNER is the first shared task on NER of species mentions and entity linking providing an exhaustively annotated large corpus of Spanish clinical case reports[5]. The objective of shared tasks is to foster the development of NLP tools and the sharing of knowledge. Our main motivation for participating was to evaluate an existing NER system and to learn from others on a shared task. In this paper, we describe the approach and the results of our participation in the task 1 of LivingNER-Species NER track (Species mention entity recognition).

IberLEF 2022, September 2022, A Coruña, Spain.

sebastien.cossin@u-bordeaux.fr (S. Cossin); gayo.diallo@u-bordeaux.fr (G. Diallo); vianney.jouhet@chu-bordeaux.fr (V. Jouhet)

0000-0002-3845-8127 (S. Cossin); 0000-0002-9799-9484 (G. Diallo); 0000-0001-5272-2265 (V. Jouhet)

Table 1
Excerpt of a tab-separated values annotation file provided by the organizers. The column filename refers to a textual case report file ; mark is an annotation identifier ; label takes two values: SPECIES or HUMAN ; off0 and off1 are the starting and ending position of the textual span in the document, respectively

filename	mark	label	off0	off1	span
caso_clinico_radiologia526	T1	SPECIES	18	21	VIH
caso_clinico_radiologia526	T2	HUMAN	0	5	Varón

2. Methods

In the following subsections, we describe the corpora, the IAMsystem algorithm and its configuration for this task.

2.1. Corpora

The dataset provided by the organizers was called the LivingNER corpus. The corpus was manually annotated by clinical experts following annotation guidelines specifically created for this task. The corpus' content was quite varied as it included annotations for animals, plants, and microorganisms and the clinical case reports came from 20 medical disciplines (cardiology, oncology, radiology...). The gold standard consisted of a collection of 2000 plain text clinical case reports written in Spanish. The LivingNER corpus was randomly sampled into three subsets: development, validation and test set composed of 1000, 500 and 500 clinical case reports, respectively. The test set was released without annotations together with a large collection of clinical case reports (background set) to avoid manual annotations. The participants had to annotate a total of 13,467 documents of which only 500 were in the test set and known by the organizers.

Each clinical case was stored as a single file in UTF8 encoding. For example, the file "caso_clinico_radiologia526.txt" starts with the following sentence: *Varón de 49 años, VIH+, al que se solicita TC abdominal urgente por sospecha de obstrucción intestinal.* The words "Varón"(male) and "VIH"(HIV) refer to a human person and a virus respectively. These terms were manually annotated by clinical experts and included in the annotation files released by the organizers. The annotation files included character offsets of entity mentions in TSV (tab-separated values) files together with their corresponding NCBI Taxonomy code annotations. An excerpt is provided in Table 1.

The participants' goal was to generate automatic annotations for the test set documents in a format similar to that of the validation and development sets (table 1).

2.2. Algorithm

IAMsystem is a biomedical semantic annotation tool developed in 2018 at Bordeaux University Hospital, France. The algorithm is a dictionary-based system, it is similar to Mgrep [6] and FlashText [7] algorithms but can handle lexical variations at the token level and single or multiword abbreviations. It was designed to efficiently annotate medical documents of a clinical

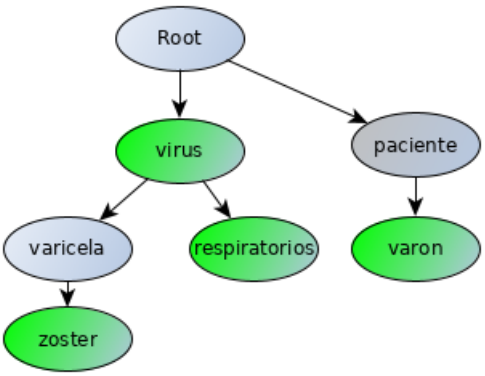

Figure 1: Tree data structure created by the algorithm to store the dictionary. In this example, the dictionary contains 4 terms: ”virus”, ”virus respiratorios”, ”virus varicela zoster”, ”paciente varon”. A green node is linked to a dictionary term while a gray node is not. The root node is a special node.

data warehouse with large terminologies such as UMLS (Unified Medical Language System) containing million of terms [8].

It takes in input a dictionary, typically in a TSV-format, and stores it in a tree data structure called a trie where each token of the dictionary corresponds to a node. A text to annotate is tokenized after undergoing the same normalization process as the dictionary: words are normalized through accents (diacritical marks) and punctuation removal, lowercasing and stopwords removal (if a stopword list is given). Given a sequence of tokens in a document, the algorithm attempts to find a path in the tree starting at the root node.

In the following example, the document starts with ”El virus de la varicela-zóster” and the dictionary is represented in figure 1. If the tokens ”el”, ”de” and ”la” are in the stopword list, they are ignored by the algorithm. At the ”virus” token, the algorithm moves from the root node to the ”virus” node, at the ”varicela” token, it moves from the ”virus” node to the ”varicela” node etc. Node transition is possible if the token of a document matches the token of a next node in the tree. Token matching is done by string matching algorithms. IAMsystem can be configured with several string matching methods to take into account typos and lexical variations. By default only exact match is performed. For example, if IAMsystem is configured with a Levenshtein distance algorithm or a soundex algorithm, it can match the token ”varicella” in a document to the node ”varicela” in the tree. IAMsystem can also handle abbreviations: if ”VVZ” stands for ”virus varicela-zóster”, it can follow the path ”root”->”virus”->”varicela”->”zoster” in the tree. If no transition is possible, the algorithm returns to the root node. The algorithm never looks for a term in the whole dictionary but for a node transition in the tree. Thus, its computational complexity depends little on the number of terms in a dictionary, which explains why it is fast for annotating a document with a large terminology. By default, the algorithm outputs the longest term detected which corresponds to the longest path in the tree. If the algorithm reaches the ”zoster” node in this example, the term ”virus varicela zoster” is returned but not the term ”virus” even though it’s also a term. A formal description of IAMsystem is provided in appendix A.

Table 2
The 10 most frequent spans with their frequency and cumulative percentage in the annotation files.

span	frequency	cumulative percentage
paciente	4861	0.209
vih	576	0.234
varón	521	0.257
personales	424	0.275
mujer	397	0.292
pacientes	292	0.305
familiares	270	0.316
madre	259	0.328
cmv	197	0.336
sars-cov-2	197	0.345

IAMsystem has already been evaluated in two shared tasks of the CLEF initiative: automatically assigning ICD-10 codes to French death certificates in 2018[9] and automatically assigning ICD-10 diagnosis and procedure codes to Spanish electronic health records in 2020[10]. The algorithm was also described on this occasion. The source code of the algorithm is available at https://github.com/scossin/IAMsystem.

2.3. Configuration for this task

The development and validation sets were combined into a single set known as the 'training set'. This training set included 1,500 clinical cases with a total of 23,203 annotations. In the span column after lowercase, these annotations had 3,418 unique values. Table 2 shows the frequency of the ten most frequent spans, as well as the cumulative percentage calculated by dividing the frequency by the total number of annotations. According to the table, approximately 21% of annotations contained the span "paciente".

Our goal was to build a dictionary that maximizes IAMsystem's F1-score on the training set. The same strategy was used in 2020 on a task of detecting symptoms and diseases mentioned in clinical notes[10]. IAMsystem used a temporary dictionary created by selecting all of the span values in the annotation files to annotate the training set. Recall and precision on the training set with this temporary dictionary were 0.97 and 0.53, respectively. IAMsystem's output file did not match perfectly the annotation files for two reasons. First of all, IAMsystem only selects the longest detected term; however, a human expert does not always make the same choice. For example, if the document contains "VIH 1" and the dictionary contains "VIH" and "VIH 1," the algorithm returns "VIH 1," even though the substring "VIH" can be annotated by humans. Second, if a human annotates a term in one document, it may not be annotated in others, whereas the algorithm will annotate it in all. Some terms had to be removed from the temporary dictionary in order to maximize the F1-score on the training set. By doing so, recall slightly decreased while precision greatly increased on the training set. To identify the terms to remove, the output file was compared to the annotation files. The frequency of each span in the annotation files and in the output file was compared: if the ratio between the two frequencies were greater than 2 the span was removed from the dictionary. For example, the span "covid-19"

Table 3
Performance of IAMsystem and average results of participants on the LivingNER test set.

System	Micro-Precision	Micro-Recall	Micro-F1 score
IAMsystem	0.9209	0.8733	0.8965
Average	0.876	0.807	0.824

appeared 231 times in the training set, resulting in 231 annotations by IAMsystem in the output file but it was annotated only a single time by human experts. Removal of this term led to the removal of one true positive and 230 false positives. A total of 109 spans were removed from the temporary dictionary resulting in a dictionary containing 3,683 terms that was used to annotate the test set. Recall and precision on the training set with this custom dictionary were 0.96 and 0.97, respectively. IAMsystem was configured without an approximate string matching method, so only exact matching method was performed.

3. Results

We submitted one run. It took about 6 seconds to annotate and generate 107,651 annotations from the 13,467 documents in the test set on a laptop with Intel Core i7-5700HQ @2.70GH x 8CPUs. Table 3 shows the performance of our system and the mean of all the submitted predictions.

4. Discussion

In 2018, IAMsystem obtained a F1-score of 0.786 (precision: 0.794, recall: 0.779) on the task of coding French death certificates with the ICD-10 terminology[9]. French death certificates consisted of short texts containing numerous typos, lexical variants and abbreviations[11]. IAMsystem was configured differently for this task.

In 2020, the same system obtained a F1-score of 0.69 (precision: 0.82, recall: 0.59) on the task of detecting symptoms and diseases mentioned in Spanish clinical notes and a F1-score of 0.52 (precision: 0.69, recall: 0.42) on the task of detecting procedures[10]. The length of the documents was comparable to this task, but the training set was smaller, with only 1.7% of the codes present [12]. It's worth noting that the same strategy to create the dictionary was used in this task, which makes comparisons easier.

For this task 1, the algorithm obtained an F1 score of 0.8965 (precision: 0.8733 , recall: 0.9209). These better results can be explained by the absence of long words dependencies in the livingNER corpus and frequent terms (table 2) that favor the micro-F1 score.

Our algorithm's main advantages are its ease of use and speed. It accepts a dictionary as input and supports a variety of fuzzy matching algorithms (Soundex, Levenshtein...). It is simple to set up and debug: the algorithm provides an explanation by displaying the start and end positions of each word sequence detected in a document containing the dictionary term. It does not require annotations to be trained, unlike machine learning algorithms. It works well when making an annotation without particular respect for the context.

Its simplicity is also the source of many of its shortcomings. A dictionary-based technique cannot detect entities that do not exist in the dictionary. The system cannot recognize a term if its words are not in the correct order or are separated in a document. It is unable to manage context and interpret the meaning of a word when necessary. We believe that new cutting-edge NLP algorithms such as BERT will outperform IAMsystem. Nonetheless, it has a role in a data scientist's toolkit for quickly establishing a baseline with a simple system.

5. Acknowledgments

This study is part of the DRUGS-SAFER research program, funded by the French Medicines Agency (Agence Nationale de Sécurité du Médicament et des Produits de Santé, ANSM). This publication represents the views of the authors and does not necessarily represent the opinion of the French Medicines Agency.

References

[1] R. Thiébaut, S. Cossin, Artificial Intelligence for Surveillance in Public Health, Yearbook of Medical Informatics 28 (2019) 232–234. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6697516/. doi:10.1055/s-0039-1677939.

[2] E. Arsevska, S. Valentin, J. Rabatel, J. de Goër de Hervé, S. Falala, R. Lancelot, M. Roche, Web monitoring of emerging animal infectious diseases integrated in the French Animal Health Epidemic Intelligence System, PloS One 13 (2018) e0199960. doi:10.1371/journal.pone.0199960.

[3] S. B. Johnson, S. Bakken, D. Dine, S. Hyun, E. Mendonça, F. Morrison, T. Bright, T. Van Vleck, J. Wrenn, P. Stetson, An Electronic Health Record Based on Structured Narrative, Journal of the American Medical Informatics Association : JAMIA 15 (2008) 54–64. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2274868/. doi:10.1197/jamia.M2131.

[4] J. Jovanović, E. Bagheri, Semantic annotation in biomedicine: the current landscape, Journal of Biomedical Semantics 8 (2017). URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5610427/. doi:10.1186/s13326-017-0153-x.

[5] Antonio Miranda-Escalada, E. Farré-Maduell, S. Lima-López, D. Estrada, L. Gascó, M. Krallinger, Mention detection, normalization & classification of species, pathogens, humans and food in clinical documents: Overview of LivingNER shared task and resources, Procesamiento del Lenguaje Natural (2022).

[6] C. Jonquet, N. H. Shah, M. A. Musen, The Open Biomedical Annotator, Summit on Translational Bioinformatics 2009 (2009) 56–60. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3041576/.

[7] V. Singh, Replace or Retrieve Keywords In Documents at Scale, 2017. URL: http://arxiv.org/abs/1711.00046. doi:10.48550/arXiv.1711.00046, number: arXiv:1711.00046 arXiv:1711.00046 [cs].

[8] O. Bodenreider, The Unified Medical Language System (UMLS): integrating biomedical terminology, Nucleic Acids Research 32 (2004) D267–270. doi:10.1093/nar/gkh061.

[9] S. Cossin, V. Jouhet, F. Mougin, G. Diallo, F. Thiessard, IAM at CLEF eHealth 2018: Concept

Annotation and Coding in French Death Certificates, arXiv:1807.03674 [cs] (2018). URL: http://arxiv.org/abs/1807.03674, arXiv: 1807.03674.
[10] S. Cossin, V. Jouhet, IAM at CLEF eHealth 2020: Concept Annotation in Spanish Electronic Health Records, CEUR workshop proceedings (2020). URL: http://www.dei.unipd.it/~ferro/CLEF-WN-Drafts/CLEF2020/.
[11] A. Névéol, A. Robert, F. Grippo, C. Morgand, C. Orsi, L. Pelikan, L. Ramadier, G. Rey, P. Zweigenbaum, CLEF eHealth 2018 Multilingual Information Extraction Task Overview: ICD10 Coding of Death Certificates in French, Hungarian and Italian, in: CLEF, 2018.
[12] A. Miranda-Escalada, A. Gonzalez-Agirre, J. Armengol-Estapé, M. Krallinger, Overview of automatic clinical coding: annotations, guidelines, and solutions for non-English clinical cases at CodiEsp track of eHealth CLEF 2020, CEUR-WS (2020).
[13] F. Pech, A. Martinez, H. Estrada, Y. Hernandez, Semantic Annotation of Unstructured Documents Using Concepts Similarity, 2017. URL: https://www.hindawi.com/journals/sp/2017/7831897/. doi:https://doi.org/10.1155/2017/7831897, iSSN: 1058-9244 Pages: e7831897 Publisher: Hindawi Volume: 2017.
[14] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages, and Computation, Addison-Wesley Publishing Company, 1979.
[15] J. Clément, P. Flajolet, B. Vallée, The Analysis of Hybrid Trie Structures, report, INRIA, 1997. URL: https://hal.inria.fr/inria-00073393.

A. Formal description

A.1. Semantic annotation

Semantic annotation consists of linking sequences of words in a document to the concepts of a terminology[13]. A terminology is composed of a set S of terms called a dictionary, a set C of concepts identified by a unique code and a relation $R \subset S \times C$ between the terms and the concepts. A document is a sequence of words $(w_1, ..., w_n)$ after a normalisation and tokenisation step. Semantic annotation consists in identifying a set of pairs $((w_i, ..., w_j), c_k)$ where $(w_i, ..., w_j)$ is a sequence of one or more words in d and c_k a concept in C. This sequence of words can be continuous or discontinuous. A sequence of words is continuous if and only if, for any integer k, $i \leq k \leq j$, w_k belongs to the sequence $(w_i, ..., w_j)$.

A.2. IAMsystem

IAMsystem is composed of a deterministic final-state automata (DFA) [14] and fuzzy string matching algorithms that are responsible of state transitions. The state diagram of the DFA forms a rooted tree, i.e. a directed acyclic graph with a single root (figure 1).

A DFA is defined by a quintuplet $(Q, \Sigma, \delta, q_0, F)$:

- a finite set of states Q
- a finite set of symbols Σ
- a transition function $\delta : Q \times \Sigma \rightarrow Q$
- an initial state $q_0 \in Q$

- a set of final states $F \subseteq Q$.

Let S be the dictionary composed of a set of terms. The tokenization function transforms a sequence of characters into a sequence of words. By applying this function to each term in the dictionary S, we obtain a dictionary S' composed of a set of word sequences. The set of symbols $\Sigma = \{w_1^r\}$ of the automata is the set of unique words of the dictionary S'. The trie associated with S' is defined recursively [15]:
$trie(S') = \{trie(S' \mid w_1), ..., trie(S' \mid w_r)\}$ where $trie(S' \mid w)$ means the subset of S' that begins with the word w. A rooted tree, denoted G, is constructed recursively from this definition.
Let G = (V,A) be a set of nodes V and edges A each composed of two nodes and a word w: $A \subset \{(x, y, w) \in V \times V \times \Sigma, x \neq y\}$. We start by creating the root of G corresponding to $trie(S')$ then, recursively, for each trie x having an element trie y beginning with the word w $\in \Sigma$, we add to the graph G a node x, a node y and an edge (x,y,w). The finite set of states of the automata Q is an alias of the set V of the graph G. Thus, each state q $\in$ Q is a node of G and the initial state q_0 is the root of G.

The transition function $\delta : Q \times \Sigma \rightarrow Q$ is defined by the set of edges A :
$\delta = \{((q1, w), q2) \mid (q1, q2, w) \in A\}$. A transition is possible between states q1 and q2 with word w if and only if there is an edge (q1,q2,w) $\in$ A. Schematically, our automata corresponds to the graph of figure 1 where each state is a node and each transition is an edge of G.

Let q_n be a node of the tree G. There is a unique path, a sequence of edges, $((q_0, q_i, w_i),(q_i, q_j, w_j), ..., (q_z, q_n, w_n))$ from the root to node q_n. By taking the 3rd element of each triplet of this sequence, we obtain a sequence of words $(w_i, w_j, ..., w_n)$. Let f : Q $\rightarrow (w_1, w_2, ..., w_n)$ be the function that maps each state $q_n \in Q$ to a sequence of words. A state q_n is a final state if and only if the sequence of words $f(q_n)$ belongs to S', i.e. if this sequence of words is the tokenisation of a term of the dictionary. $F \subset Q$ is the set of final states defined by $F = \{q \mid q \in Q, f(q) \in S'\}$

Let Σ^* be the infinite set of word sequences that can be formed from the set of symbols Σ. The function $next : Q \times \Sigma^* \rightarrow Q$ searches a state transition in the automata with a sequence of words. The function takes as inputs a state of the automata $q_1 \in Q$ and a sequence of words $(w_1, w_2, ..., w_n) \in \Sigma^*$ and returns a state $q_2 \in Q, q_2 \neq q_1$ if there is a transition (a path) between q_1 and q_2 via the supplied word sequence. This function *next* uses the function δ for each intermediate transition between two states. If no state transition is possible, the function returns $\emptyset \in$ Q.

For each word w_{input} in the document, a set of word sequences is produced by each of the fuzzy string matching functions. Let g be a fuzzy string matching function, $g : w_{input} \rightarrow \{seq \mid seq \in \Sigma^*\}$. A function g receives as input a word w_{input}, which may or may not belong to the set of symbols Σ, and returns a set of sequences of words where each word belongs to Σ.

For each word (token) in the document, each fuzzy string matching function is called and their results are grouped into a set $SEQ \subset \Sigma^* = \bigcup_1^n g(w_{input})$. For each seq $\in$ SEQ, the function *next* is called to search for a state transition. If a transition is found, the automata changes its state. If the state is a final state then a term in the terminology has been detected and a semantic annotation is produced. If multiple transitions are possible, IAMsystem duplicates the automata to explore several paths in the tree and can therefore contain several copies of the automata. Although each automata is in a unique state, the IAMsystem algorithm can be located in several different states. If no transition is found, the automata returns to the initial state q_0 and starts

again. If no transition is found in the initial state, the algorithm moves to the next token in the document.

The IAMsystem algorithm is described in pseudocode below.

Algorithm 1 IAMsytem

Input: a document d,
a final-state automata $\mathscr{A}$,
functions next and tokenize
fuzzy string matching algorithms g_i
Output: a sequence of annotations (Annots)

```
Annots = []
doc_tokens := tokenize(d)
states := [q_0]
w_input := first token in doc_tokens
while w_input ≠ e do
    if w_input is a stopword then
        next
    end if
    SEQ = ⋃_{i=1}^{n} g_i(w_input)
    for seq in SEQ do
        new_states := []
        for state in states do
            new_state := next(state, seq)
            add new_state to new_states
        end for
    end for
    if length new_states = 0 then
        for state in states do
            if state ∈ F then
                create an annotation and add it to Annots
            end if
        end for
        if states = [q_0] then
            w_input := next token
        else
            states := [q_0]
        end if
    else
        states := new_states
        w_input := next token
    end if
end while
return Annots
```