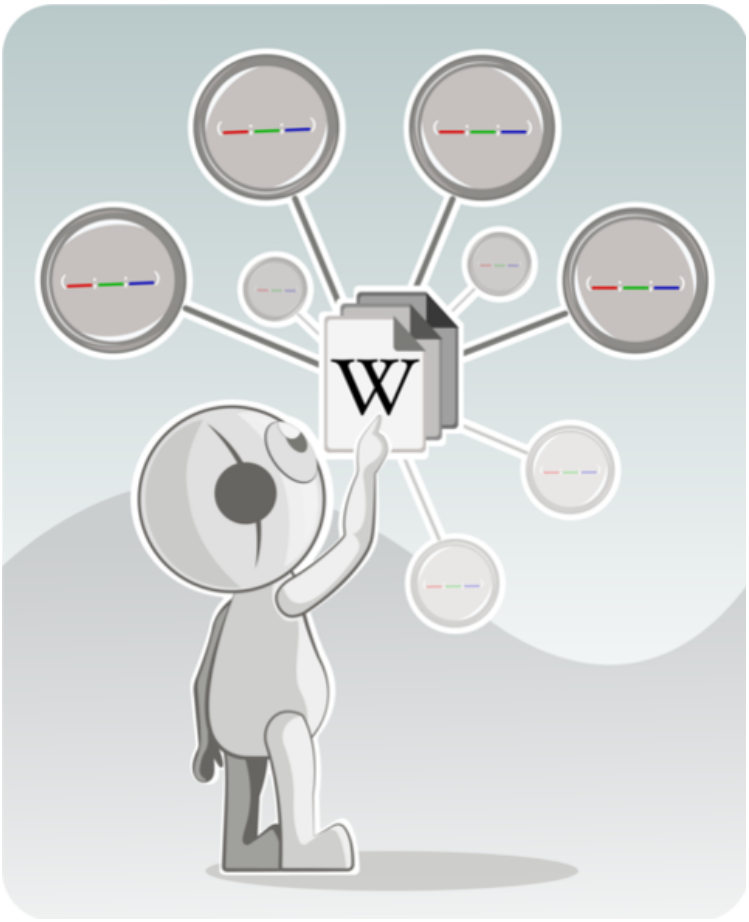
Semantic extraction - NLP
Group 9 Report



Università degli studi di Salerno

2021/2022

Valentino Daniele	0622701069
D'Amato Giuseppina	0622701577
Ciraudo Egidio	0622701566
Aleksandruk Lyubov	0622701472

GitHub Link

Summary

Study of the problem - Data Analysis

Automating the structuring of the text has become a problem of great interest, especially with the recent spread of Big Data.

The idea of this project is to take a (more or less complicated) text and, through text processing and all NLP activities, extract subject-predicate-object triples.

Once we found the triples, we used DBPedia for the URI associated with subject and object and WordNet to find the meaning of the predicate.
In particular, we focused on the significance of the textual content, in order to obtain triples that were as representative as possible of the meaning of the text.

Goal Example:

```
Harry ability          speak          snake language Parseltongue

URI:
Harry Potter - http://dbpedia.org/resource/Harry_Potter
snake - http://dbpedia.org/resource/Snake
Parseltongue - http://dbpedia.org/resource/Magic_in_Harry_Potter

VERB SENSES:
speak - verb: express in speech
```

The text we worked on consists of several paragraphs, more or less long sentences, with subordinates of various types. We consider as useful paragraphs only the ones ending with a point.
It is necessary to take into consideration the characteristics of the text because they influence some design choices.
The text contains many **co-references** that occur when two or more expressions in a text refer to the same person or thing; they have the same referent.

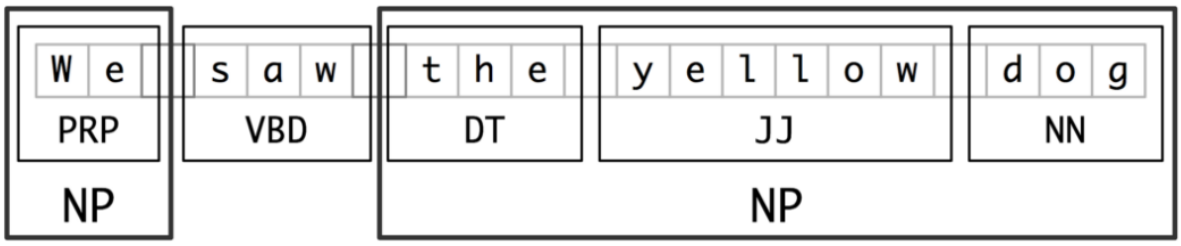
Example:

`The central character in the series is Harry Potter, a boy who lives in the fictional town of Little Whinging, Surrey with his aunt, uncle, and cousin – the Dursleys.`

→ The proper noun **Harry Potter** and the pronoun **'his'** refer to the same person, namely to Harry Potter.

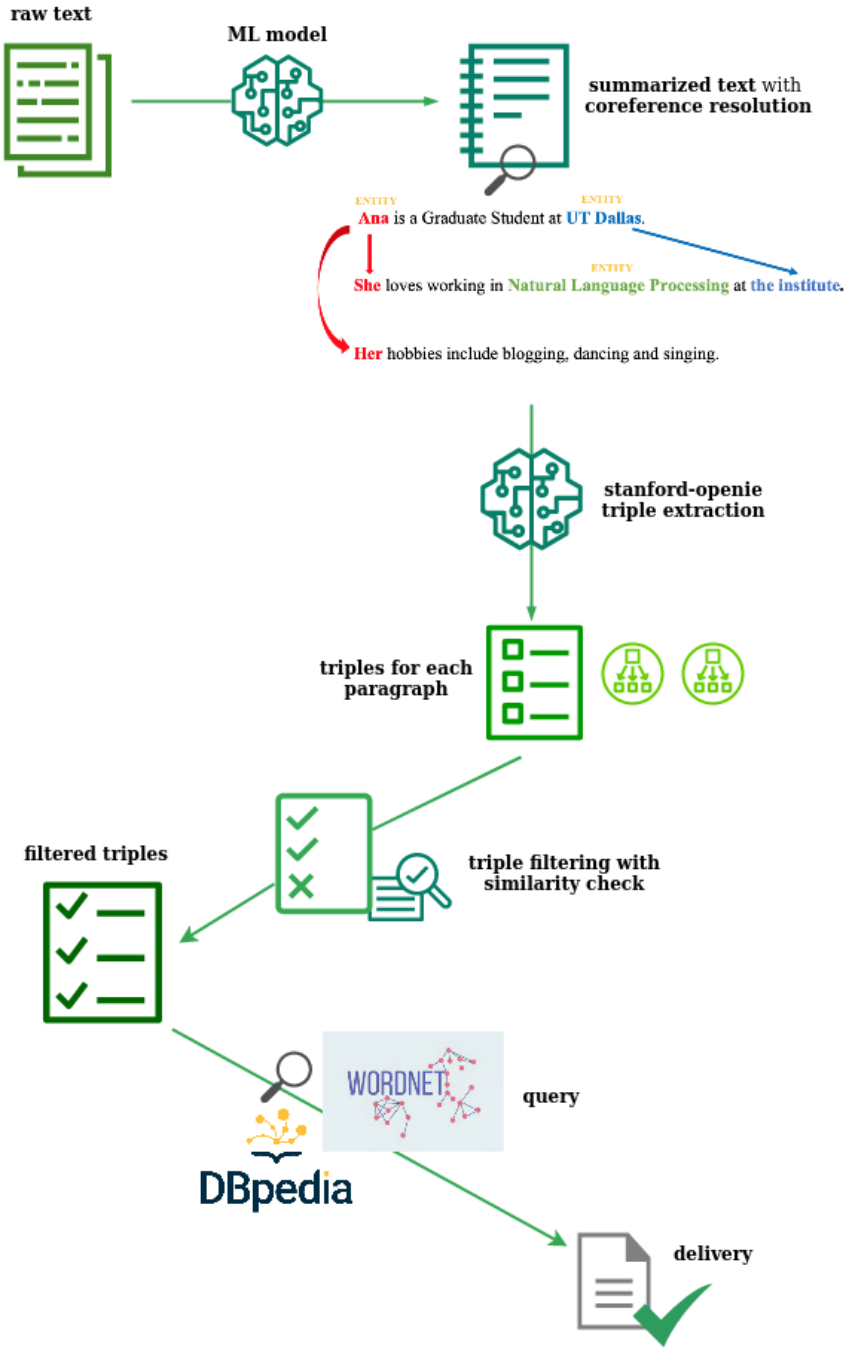
We have chosen to do the following elaborations on the text:

- *synthesis*: in order to extract only the most relevant information, this is particularly useful when dealing with long texts and complex phrases;
- *removal of stopwords*: a stop word is a commonly used word such as "the", "a", "an", "in". We tried to eliminate the stopwords with NLTK, which has a list of stopwords stored in 16 different languages, but this elimination involved an excessive loss of information content in the extracted triples. For this reason we have removed them, by storing a list of words that we consider as stop words. This is how we are making our processed content more efficient by removing words that do not contribute to any future operations. The elimination of the stopwords was done in two different phases: before triples extraction, we removed a set of stopwords (– , ` ` , '', 's, '), while **after the extraction of the triples** we removed another set of stopwords (as, a, also, As, the , The , Against, against) in order to better perform queries on DBpedia and WordNet.
- *lemmatization:* it is the process of converting a word to its base form. It was done after the extraction of the triples.
- *chunking:* it is a process of extracting phrases from unstructured text, which means analyzing a sentence to identify the constituents (noun groups, verbs, verb groups, etc.). For example, the solution for the extraction of triples based on patterns starting from the Syntax tree was excluded, after being tested, due to the difficulty in being able to generalize long and complex sentences with a single pattern. We tried this solution but **it is not** in the final deliverable.



Chunking: Phrases divided into chunks(Source:https://www.nltk.org)

System overview/architecture



Description of the pipeline

The input of our system is a raw text, consisting of several paragraphs, each containing several sentences.
Our first step was to summarize the text using the **BART** transformer, which allows us to obtain more synthetic sentences that respect the subject-predicate-object pattern.
On the new text obtained, we apply the coreference resolution with **Neuralcoref** to find all the expressions in the text that refer to a specific entity. This will allow us to make later queries to DBpedia that are more meaningful as they will always have an entity as subject and not a pronoun.
We now extract the subject-predicate-object triples for each paragraph of the text using a Python version of **Stanford OpenIE**. We noticed that the triples extracted, belonging to the paragraph, were very similar to each other, i.e. they expressed the same concept.
We therefore decided to filter triples on each paragraph in this way:

- using **paraphrase-MiniLM-L6-v2** we calculate the embeddings of the paragraph and the list of triples (in the form of strings) extracted from this paragraph; in the context of neural networks, embeddings are low-dimensional, learned continuous vector representations of discrete variables;
- we calculate the **cosine similarity** of the embedding of each triple with respect to the embedding of the paragraph from which it was extracted; the cosine similarity measures the cosine of the angle between two vectors projected in a multi-dimensional space.
- an **average** of the cosine similarities of the triples for each paragraph is calculated;
- only the triples whose cosine similarity value is greater than or equal to the previously calculated average are selected.

This average calculation is necessary because we noticed that if we set a static threshold value, for example 0.7, for some paragraphs not a single triple was selected, since the similarity values of the triples were all below this value despite the fact that there were significant triples to be taken into consideration.
By filtering the queries with the similarity function, we significantly decreased the number of triples: starting from 410 triples, we got 303 triples because most of them were very similar to each other.
Once the filtered triples have been obtained, for each of them we make queries to **DBpedia** to extract the URIs relating to the entities contained in the subject and object.
Finally, with regard to the predicates of triples, we generally expect them to be composed

of verbs, so we search for their meaning in the **WordNet** database. If adverbs, prepositions or nouns are present in a predicate, their WordNet definition will be returned.

These are the libraries we used:

BART (large-sized model), fine-tuned on CNN Daily Mail

We have used the BART model pre-trained on English language, and fine-tuned on CNN Daily Mail. It was introduced in the paper BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension by Lewis et al. and first released in this repository .
BART is a transformer encoder-decoder (seq2seq) model with a bidirectional (BERT-like) encoder and an autoregressive (GPT-like) decoder. BART is pre-trained by (1) corrupting text with an arbitrary noising function, and (2) learning a model to reconstruct the original text. The model is particularly effective when fine-tuned for text generation (e.g. **summarization**, translation) but also works well for comprehension tasks (e.g. text classification, question answering). This particular checkpoint has been fine-tuned on CNN Daily Mail, a large collection of text-summary pairs.

Coreference resolution with Neuralcoref

The coreference resolution tries to find all expressions in the text that refer to a specific entity. We have used the Neuralcoref model from Huggingface that runs on top of the SpaCy framework. NeuralCoref is a pipeline extension for spaCy 2.1+ which annotates and resolves coreference clusters using a neural network.
NeuralCoref is production-ready, integrated in spaCy's NLP pipeline and extensible to new training datasets. We have used the default parameters of the Neuralcoref model.

paraphrase-MiniLM-L6-v2

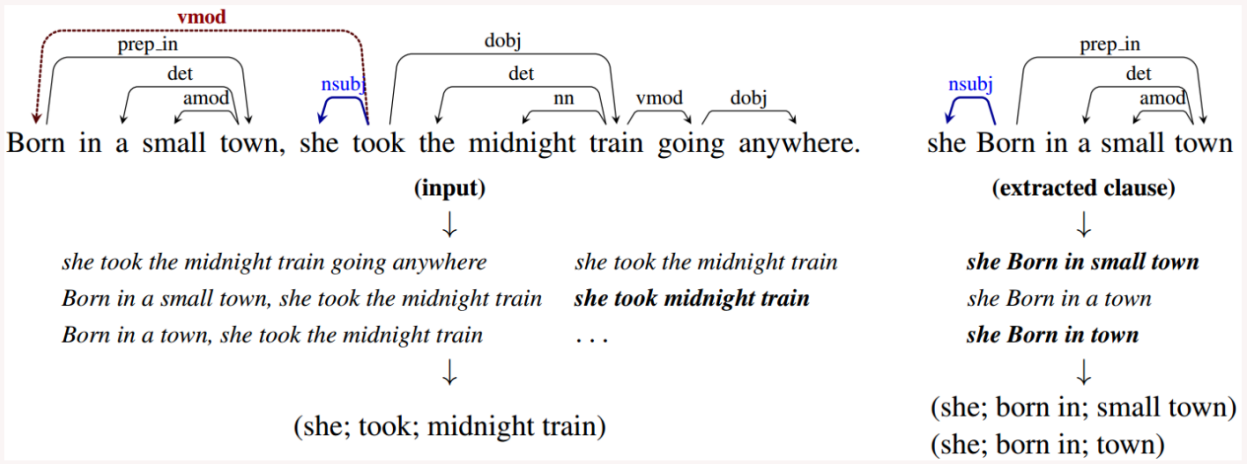
This is a sentence-transformers model: it maps sentences & paragraphs to a 384 dimensional dense vector space and can be used for tasks like clustering or semantic search. SentenceTransformers is a Python framework for state-of-the-art sentence, text and image embeddings. The initial work is described in the paper Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. You can use this framework to compute sentence / text embeddings for more than 100 languages. These embeddings can then be compared e.g. with cosine-similarity to find sentences with a similar meaning. This can be useful for semantic textual similar, semantic search, or paraphrase mining.

Python3 wrapper for Stanford OpenIE

GitHub - philipperemy/stanford-openie-python: Stanford Open Information Extraction made simple! is a python wrapper for Stanford OpenIE that is a Java implementation of an open IE system described in the paper Leveraging Linguistic Structure For Open Domain Information Extraction.

Open information extraction (open IE) refers to the extraction of relation tuples, typically binary relations, from plain text, such as (Mark Zuckerberg; founded; Facebook). The central difference from other information extraction is that the schema for these relations does not need to be specified in advance; typically the relation name is just the text linking two arguments. For example, Barack Obama was born in Hawaii would create a triple (Barack Obama; was born in; Hawaii), corresponding to the open domain relation was-born-in(Barack-Obama, Hawaii).

The system first splits each sentence into a set of entailed clauses. Each clause is then maximally shortened, producing a set of entailed shorter sentence fragments. These fragments are then segmented into OpenIE triples, and output by the system. An illustration of the process is given for an example sentence below:



DBpedia Spotlight

DBpedia Spotlight is a tool for automatically annotating mentions of DBpedia resources in text. Improving Efficiency and Accuracy in Multilingual Entity Extraction approach https://github.com/dbpedia-spotlight/dbpedia-spotlight-model.

Implementation

The idea behind the development of the systems is to use 3 separate main modules:

1. Summarization module.
2. Triples extraction module.
3. Query module.

Summarization

summarizer.py - In this file, it is taken as input a textual file, namely *HP-Wikipedia.txt* with the paragraphs to analyze. We noticed in the given text that the paragraphs all ended with a point, while titles (which brought no useful content for us), had no point at the end. So, as a design choice, we decided to consider as useful paragraphs only the ones ending with a point.
After reading the text, it is applied summarization. Then, on the obtained summary it is applied coreference resolution. Finally, a textual file with the result of the operations done in this module is generated: *summary.txt* .

Triples Extraction

triples_extraction.py - In this file, it is taken as input a textual file (summary.txt), containing the summarized paragraphs of the original text. Using StanfordOpenIE library every paragraph is then analyzed, in order to extract all the triples. Then, for each paragraph, the triples are filtered using cosine-similarity, in order to extract only those ones that represent the most the original meaning of the text. Then we apply lemmatization on every triple and remove our custom set of stop-words. Finally, two textual files with the result of the operations done in this module are generated: triples.txt (which contains every triple found) and *filtered_triples_avg.txt* (which contains the triples obtained after filtering).

Query

query.py - In this file, it is taken as input a textual file (filtered_triples_avg.txt), which contains the triples we want to analyze. For every triple, subject, predicate and object are taken separately. On subject and object, we apply two different types of query. First, we use Spacy DBpedia Spotlight and then we use a sparql query written by us. Then, we combine the two results. For predicates we use WordNet to extract verb senses. Finally, all the obtained results are combined in a textual file output (final_triples.txt).

We didn’t provide a graphical user interface, but it could be a goal for further implementations in the future.

Validation and testing

Summarization

In order to synthesize the text, two different network models have been tested, below are shown the outputs of both with reference to the original text.

- original text:
    - *"An epilogue Nineteen Years Later describes the lives of the surviving characters and the effects of Voldemort's death on the Wizarding World. In the epilogue, Harry and Ginny are married with three children, and Ron and Hermione are married with two children."*
- output with ***sshleifer/distilbart-cnn-6-6:***
    - *"In the epilogue, Harry and Ginny are married with three children, and Ron and Hermione* ***are married to two children*** *. The epilog describes the lives of the surviving characters ."*
- output with ***facebook/bart-large-cnn:***
    - *"An epilogue Nineteen Years Later describes the lives of the surviving characters and the effects of Voldemort's death on the Wizarding World. In the Epilogue, Harry and Ginny are married with three children, and Ron and Hermione are married."*

As you can see, the first synthesis is much shorter and more compact, but is generated a sentence absolutely meaningless and ambiguous that would have compromised the subsequent extraction of triples, while the second model removes this incorrect information so it is preferable the second solution in situations like this. It must be specified that the choice of a model rather than another, was dictated by the analysis of the overall synthesis of the various paragraphs. Both models in fact commit errors in the synthesis, often in reality, as in the sentence in the example above, these errors are due to the ambiguity of the sentences of departure.

Stemming vs Lemmatization and stop-words removal

The aim of both processes is the same: reducing the inflectional forms of each word into a common base or root.

Stemming algorithms work by cutting off the end or the beginning of the word, taking into account a list of common prefixes and suffixes that can be found in an inflected word.

Lemmatization, on the other hand, takes into consideration the morphological analysis of the words. To do so, it is necessary to have detailed dictionaries (such as WordNet) which the algorithm can look through to link the form back to its lemma.

We used NLTK's stemmer, Spacy's lemmatizer and NLTK's lemmatizer which uses WordNet internally, below are the results obtained for comparison.

Stemmed version of "*The central character in the series is Harry Potter, a boy who lives in the fictional town of Little Whinging, Surrey with his aunt, uncle, and cousin – the Dursleys – and discovers at the age…"* becomes "*central charact seri harri potter boy live fiction town littl whing aunt cousin dursley discov age…".*

As you can see, the words are completely changed making it impossible to recognize important information such as Harry Potter, so this solution was ruled out immediately.
Using the lemmatizer provided significantly better results, however there are some differences in how the two algorithms work.
For instance with Spacy's Lemmatizer words *"wizarding academy"* become *"wizarde academy"* so in general we have preferred the lemmatizer of NLTK.
In addition, through the use of the lemmatizer all plural words have been returned to the singular form, all verb tenses have been returned to the base form, e.g. 'are', 'is', 'was', 'will' become 'be'.
We can say that stemming is a quick and dirty method of chopping off words to its root form while on the other hand, lemmatization is an intelligent operation that uses dictionaries which are created by in-depth linguistic knowledge. Hence, Lemmatization helps in forming better features.

In order to be able to extract enough meaningful triples, we thought of removing words that appear often in the text, but do not make a significant contribution to the information such as temporal information, conjunctions and so on.
We initially tried removing stopwords from the text before extracting triples, but this often resulted in the extraction of triples that were not very meaningful. Such triples in fact were initially extracted from a tree generated from tagging individual words in the text. To extract the triples a pattern was used that went to look for branches of the tree with NP tags for the subject and VP branches for verb and object within it, but the entire structure of the tree in the presence or absence of stopwords was significantly modified.
We used the English language stopword set and after a series of tests we realized that:
without the removal of the stopwords, among the triples that were extracted there was 'Harry Potter be centrer character', this triple was lost instead with the removal of the

stopwords as within the set there is the verb 'be'. Another observation we could make is that often in the text there are compound verbs such as 'look for', so the removal of the word 'for' completely changes the sense of the sentence.
A study also showed that the removal of stopwords is more appropriate for text classification problems and not for summarization problems, so in our final solution, we decided not to apply the removal of all stopwords in order not to lose too much information content, but to remove only a few words or symbols such as quotation marks, hyphens and brackets on the extracted triples.

Coreference resolution

As we analyzed the text, we noticed that there were more complex sentences than others in which the subject was at the beginning of the text, and there were many subordinates in which we referred to the subject by the personal or possessive pronoun. Extracting the triples on the raw text, synthesized or not, it was identified as subject 'he', 'it' etc... that without a context did not contribute any significant information.

To solve this problem, we thought to use the coreference resolution, that is we identified all references to the same subject and replaced them with the subject itself, for example every time in the text appear the words 'he', 'him', 'his' ... which refer to Harry Potter, are replaced with 'Harry Potter'.

However, we found some problems in the coreference resolution made in particular by Nauralcoref: 'The central character of the series is Harry Potter, a boy who lives in the fictional town of Little Whinging, Surrey, with his aunt, uncle and cousin - the Dursleys - and discovers at the age of eleven that he is a wizard, even though he lives in the ordinary world of non-magical people known as Muggles.' the subject is given by the central character of the series is Harry Potter so all of these are replaced by 'his' and 'he'. On one hand, long spans capture the context and tell us more about the real-world entity we're looking for. On the other hand, they often include too much information.

We would not replace each pronoun with such an extended expression - especially in the case of nested spans.

We then noticed that from the synthesized text, the first sentence was being transformed into 'Harry Potter is a boy who lives in the fictional town of Little Whinging, Surrey with his aunt, uncle and cousin - the Dursleys - and discovers at age 11 that he is a wizard. ' so through the same coreference resolution function, the same pronouns were simply replaced with Harry Potter:

'*Harry Potter is a boy who lives in the fictional town of Little Whinging, Surrey with Harry Potter aunt, uncle and cousin - the Dursleys - and discovers at the age of 11 that Harry Potter is a wizard.'*

Another aspect that emerged was the substitution of possessive pronouns, for example 'Harry lives with ***his*** uncle' should have become 'Harry lives with ***Harry's*** uncle', but Neuralcoref is not able to make such a precise substitution unlike other models that do coreference resolution and furthermore this did not discriminate the detection of good triples.

DBpedia query

In order to extract the URIs relating to the entities contained in the subject and object of the various extracted triples, we made SPARQL queries to DBpedia Ontology, going to search for these entities in the English language.
However, we noticed that sometimes this approach did not bring the desired results: parallel to the SPARQL queries, we searched for the same entities on DBpedia Lookup and in some cases a result was returned here.
For this reason we have decided to combine two approaches for the extraction of URIs: the query written by us and the use of Spacy DBpedia Spotlight, this package acts as a Entity Recogniser and Linker using DBpedia Spotlight, annotating SpaCy's Spans and adding them to the entities annotations.
Using this second solution we noticed that some curious situations, for example 'Tom Marvolo Riddle' is recognized as 'Lord Voldemort', so both 'Tom Marvolo Riddle' and 'Lord Voldemort' have the same URI; 'Triwizard Tournament' is recognized as one of the books of the series: 'Harry Potter and the Goblet of Fire', as well as 'Deathly Hallows' which is identified with 'Harry Potter and the Deathly Hallows'; 'Invisibility Cloak' which is among 'magical objects in Harry Potter'.

Example of filtered triples

These are triples obtained **initially**:

{*'subject': 'wizarding world', 'relation': 'contains', 'object': 'versions of elements of everyday life'*}

{*'subject': 'wizarding world', 'relation': 'contains', 'object': 'magical versions of elements of everyday life'*}

{*'subject': 'wizarding world', 'relation': 'contains', 'object': 'versions of elements'*}

{*'subject': 'wizarding world', 'relation': 'contains', 'object': 'versions'*}

{*'subject': 'wizarding world', 'relation': 'contains', 'object': 'magical versions of elements'*}

{*'subject': 'wizarding world', 'relation': 'contains', 'object': 'versions of ordinary elements of life'*}

After assessing the cosine similarity between these triples and the paragraph from which they were extracted, we only considered the triples that were 'closest' to the paragraph:

{*'subject': 'wizarding world', 'relation': 'contain', 'object': 'version of element of life'*}

{*'subject': 'wizarding world', 'relation': 'contain', 'object': 'magical version of ordinary element of life'*}