CMPUT 690: Assignment 2

Peng Xu, *pxu4@ualberta.ca*

1. Preprocess

- Split the original document into 826 separate documents, which can be found in the folder named with "data".
- Replace '\\n' with '\n' in the original text.
- Source code can be found in 'src/split.py'.

2. Stage 1: Name Entity Recognition

Via NLTK package in python, we can easily access Stanford NER Tagger. After tokenizing the documents, we can get the name entity recognition result with the tagger. With a little manipulation, we can get name entities with three classes: PERSON, ORGANIZATION, and LOCATION. And we will extract the relations between these name entities.

Because these name entities commonly are word phrases, which consist of several words, I create a mapping from name entities to a single word. For example, "Antoine Griezmann" is indicated by "PERSON0" and "Real Sociedad" is indicated by "ORGANIZATION1". The mapping relations are stored in 'src/entity.txt'. Meanwhile, I replace the name entities with the indicators in the separate documents, in order to make the subsequent process more convenient. All these transformed documents can be found in the folder named with "data". The source code can be found in 'src/ner.py'.

3. Stage 2: Extract Raw Relations via Dependency Tree

Following the idea in "PATTY: A Taxonomy of Relational Patterns with Semantic Types", we first apply the Stanford Dependency Parser to the individual sentences of the documents to obtain dependency paths. The dependency paths form a directed tree, with words being nodes and dependencies being edges.

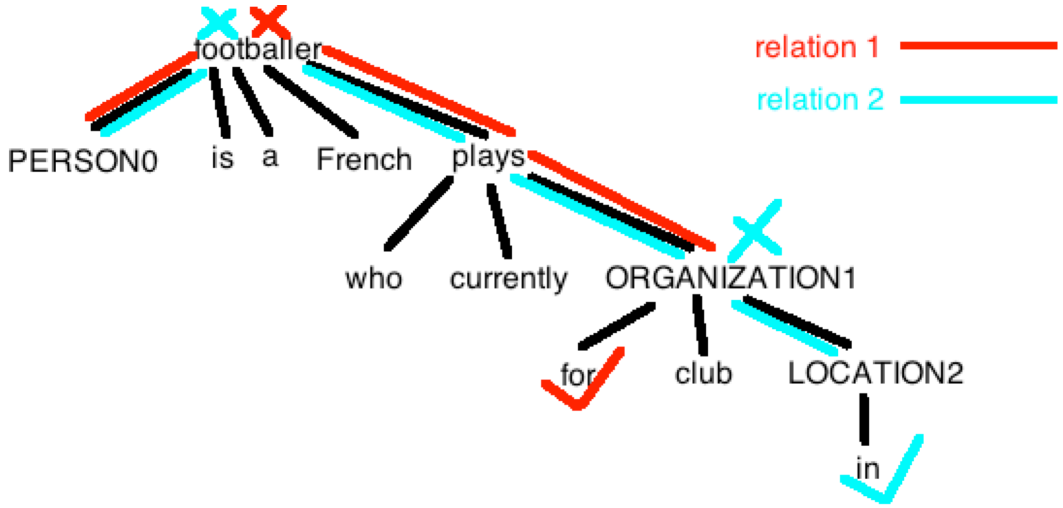
Using the name entities detected in the previous stage, we can now try to extract the relations. Whenever two name entities appear in the same sentence, we extract a raw relation. For this purpose, we traverse the dependency tree to get the shortest path that connects the two entities. Because of the tree structure, we just need to find the lowest common ancestor (LCA) of two entities, and then we can find the shortest path between two entities.

Due to the intricacy of the sentence structure, there are many tricks that needed to be considered carefully during the implementation. First, we should carefully treat

duplicate words in the sentence to avoid cycles in our dependency tree. Our LCA algorithm will be stuck in the cycles and never get a proper result. Second, sometimes the relations between two entities are somewhat unclear or too complex, we need to add some constraints to filter these relations. We remove the nouns along the paths. Furthermore, we only allow one verb existing along the paths. Third, we expand the shortest path with preposition or subordinating conjunction (denoted as 'IN' in Penn Treebank P.O.S. Tags) modifiers like 'for', 'in' and 'of'. After above work, we extract some relations over the whole documents. But it seems there are quite a lot unreasonable relations extracted.

After looking into the extracted relations, I find that most reasonable relations are in the first sentence of the documents. Then I decide to focus on the first sentence of each document. Meanwhile, I find the brackets, which usually appear in the first sentences, often effect the extraction. So if there are brackets in the first sentence, I will treat it as two different sentences (same subject with different contents). Eventually, I extract the raw relations.

The figure below illustrates an example how we extract the relations via dependency trees. The origin sentence is "PERSON0 is a French footballer who currently plays for ORGANIZATION1 in LOCATION2". The dependency tree is denoted by black line in the figure. After the extraction, we obtain two raw relations: **PERSON0 plays for ORGANIZATION1** (denoted by red line); **PERSON0 plays in LOCATION2** (denoted by blue line).



The extracted relations can be found in 'output/raw.tsv'. The source code can be found in 'src/extraction.py'.

4. Stage 3: Cluster the Raw Relations into Formal Relations

After some observations on the raw relations extracted, I find that there are mainly several kinds of formal relations: **play for**, **play in**, **refer as**, **born in**, **locate in**, **found by**, **designed by**. There are various patterns for each of the general relations. Because the total number of relations is limited, it's not that hard to set rules to cluster the raw relations. In the clustering process, I also abandon some poor-structured and meaningless raw relations.

In the clustering process, I also count the frequency of each pattern. The final clustering relations can be found in 'output/cmput690w16a2.tsv' and the source code can be found in 'src/cluster.py'.

5. Bonus: Most Effective Relational Patterns for each Relation

- **play for:** *associates people to organizations they played for as a footballer*
  - [PERSON] play for [ORGANIZATION] (**Frequency: 613**)
- **play in:** *associates people to locations where they played in as a footballer*
  - [PERSON] play in [LOCATION] (**Frequency: 148**)
  - [PERSON] play from [LOCATION] (**Frequency: 194**)
- **refer as:** *associates two co-reference entities*
  - [ENTITY] refer as [ENTITY] (**Frequency: 46**)
  - [ENTITY] simply [ENTITY] (**Frequency: 28**)
  - [ENTITY] known as [ENTITY] (**Frequency: 116**)
  - [ENTITY] abbreviated to [ENTITY] (**Frequency: 1**)
  - [PERSON] nicknamed [PERSON] (**Frequency: 2**)
- **born in:** *associates people to locations where they were born in*
  - [PERSON] born in [LOCATION] (**Frequency: 381**)
- **locate in:** *associates organizations or locations to locations where they located in*
  - [ORGANIZATION] located in [LOCATION] (**Frequency: 30**)
  - [ORGANIZATION] based in [LOCATION] (**Frequency: 9**)
  - [LOCATION] in [LOCATION] (**Frequency: 343**)
- **found by:** *associates organizations to people who found them*
  - [ORGANIZATION] found by [PERSON] (**Frequency: 2**)
- **designed by:** *associates locations to people who designed them*
  - [LOCATION] designed by [PERSON] (**Frequency: 1**)

6. Issues Encountered

- My methods cannot handle very complicate sentence, especially there are a lot name entities in it. However, this case appears frequently in this corpus.
- Some extracted relations is unreasonable because the subject and the object are too far away from each other.

- When clustering, I just set the rules manually, which cannot generalize well to other types of corpus. The papers in the reading list mention a lot approaches to cluster relational patterns automatically. I'm especially interested in the LDA-style approach, and I will try on it later on if possible.

7. Appendix

Statistics on Final Results

- 7 types of formal relations, 14 types of relational patterns
- 1887 relations extracted in total