L'esercitazione prevede l'implementazione di un sistema di OIE (lezione 5 giugno)

5.2 Svolgimento

L'open information extraction consiste in una serie di tecniche che permettono di estrarre da frasi delle informazioni Il procedimento implementato estrae tre tipi di informazione, una basata sui patterns descritti da Hearst (il pattern *such as*, che va a definire l'iperonimia), una che descrive per ogni verbal phrase all'interno della frase gli argomenti ed infine l'ultima che va ad evidenziare le adjectival phrases.

Hearst pattern	Example occurrences
X and other Y	...temples, treasuries, and other important civic buildings.
X or other Y	bruises, wounds, broken bones or other injuries...
Y such as X	The bow lute, such as the Bambara ndang...
such Y as X	...such authors as Herrick, Goldsmith, and Shakespeare.
Y including X	...common-law countries, including Canada and England...
Y, especially X	European countries, especially France, England, and Spain...

Tabella che descrive i pattern trovati da Hearst

Inizialmente le frasi vengono estratte dal SEMCOR corpus. Per ogni frase viene calcolato l'albero a dipendenze grazie alla risorsa spaCy e come prima cosa si utilizza la funzione Matcher di spaCy (che permette di controllare che in una frase ci sia un determinato pattern) proprio sul pattern "*such as*" :

```
such_pattern = [
                {'POS':'NOUN'},
                {'LOWER': 'such'},
                {'LOWER': 'as'},
                {'POS': 'PROPN'} #proper noun
                ]
```

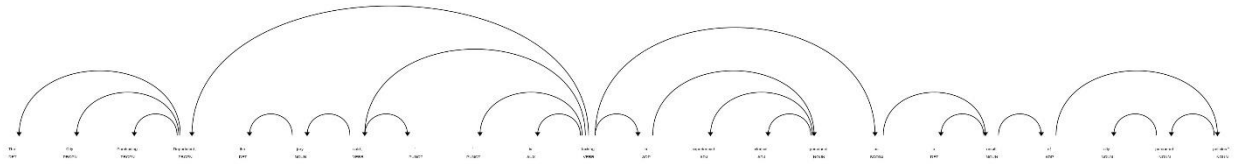
E In seguito si va ad estrarre ogni verbo ed i suoi rispettivi argomenti dalle frasi utilizzando la funzione `getVerbRelations()`, la quale naviga l'albero a dipendenze creato da spaCy per poi andare a prendere l'elemento di tipo *nsubj* e *nsubjpass*, che abbia come POS *N* oppure *PropN*, oppure *Pron* , che si trovi a destra del token di tipo *verb* e l'elemento di tipo *dobj*, quindi il complemento oggetto, se presente. Per ogni frase restituisce un tripla *subj, verb, obj*, se è presente il complemento oggetto, oppure una tupla *subj, verb*. Di seguito l'implementazione:

```
def getVerbRelations(text, nlp):
    doc = nlp(text)
    sent = []
    for token in doc:
```

```
        # if the token is a verb
        if (token.pos_=='VERB'):
            phrase =''
            # only extract noun or pronoun subjects
            for sub_tok in token.lefts:
                if (sub_tok.dep_ in ['nsubj','nsubjpass']) and (sub_tok.pos_ in
['NOUN','PROPN','PRON']):
                    # add subject to the phrase
                    phrase += sub_tok.text
                    # save the root of the verb in phrase
                    phrase += ', '+token.lemma_
                    # check for noun or pronoun direct objects
                    for sub_tok in token.rights:
                        # save the object in the phrase
                        if (sub_tok.dep_ in ['dobj']) and (sub_tok.pos_ in
['NOUN','PROPN']):
                            phrase += ', '+sub_tok.text
            sent.append(phrase)
    sent = [s for s in sent if s]
    return sent
```

La funzione `getAdjectiveRelations ()`, in maniera simile alla funzione appena descritta esplora l'albero della frase, andando però a prendere le relazioni di tipo aggettivale e restituendo delle tuple del tipo *noun, adjective*.

Di seguito un esempio di albero a dipendenze ed un piccolo estratto dei risultati:



Albero della frase "The City Purchasing Department , the jury said , `` is lacking in experienced clerical personnel as a result of city personnel policies '' ."

Sentence	Hypernym relations	Verbal relations	Adjective relations
The Fulton County Grand Jury said Friday an investigation of Atlanta 's recent primary election produced `` no evidence '' that any irregularities took place.	[]	['Jury, say', 'irregularities, take, place']	['recent primary election']