Web Ontology Exploration from Text

In this section you will do the "Hello World" of ontology learning: selecting significant terms and exploring their relations from web knowledge.

Objectives

By the time you complete this section you will:

- Understand how to pre-process the text before further training
- Use the spaCy API to deal with textual information and prepare terms for training
- Create a simple feature representation to perform term clustering
- Select the significant terms from term clusters
- Use Sparql query to catch knowledge of from a knowledge base DBpedia
- Observe the performance of knowledge acquisition

The Problem: Relation Discovery from External Knowledge Base

Relation discovery from external knowledge base, which asks a program to acquire the relation knowledge of a certain term or acquire more terms connecting by a certain term.

The Solution: SPARQL query

SPARQL query excels at solving problems where the program needs to query to a knowledge base. By indicating the query subject, and extracting the knowledge from the returns, the relation knowledge could be discovered.

The Music Dataset

The provided Music Corpus is a concatenation of several music-specific corpora, i.e., music biographies from Last.fm contained in ELMD 2.0 , the music branch from Wikipedia, and a corpus of album customer reviews from Amazon. Specifically, it is the 100M-word corpus including Amazon reviews, music biographies and Wikipedia pages about theory and music genres.

Here we have random selected 1000 documents for experiments, which is stored in df_select_1000doc.csv.

Loading the required packages

[1]:
```
from collections import Counter
import numpy as np
import pandas as pd
import os
import spacy
from spacy.lang.en.stop_words import STOP_WORDS #| version>2.0
import requests
from gensim.models import Word2Vec
from sklearn import cluster

# pip install SPARQLWrapper
from SPARQLWrapper import SPARQLWrapper, JSON
```

[2]:
```
# load music corpus
path_here = os.getcwd()
df_select_doc = pd.read_csv(path_here +'/df_select_1000doc.csv')
```

Pre-processing text into noun phrases(NPs)

[13]:
```
# recognize those NPs (require many time for 1000 docs, here we use only the top 100 docs to shorten execution time)
nps_doc = []
nlp = spacy.load('en_core_web_sm')

for text in df_select_doc['text'][:100]:
    np_doc = []
    doc = nlp(text)
    for chunk in doc.noun_chunks:
        np_doc.append((chunk.text, chunk.root.text, chunk.root.dep_))
    nps_doc.append(np_doc)
```

[14]:
```
# acquire the original NPs
source_nps_doc = []

for np_doc in nps_doc:
    tmp = []
    for np1, root, dep in np_doc:
        tmp.append(np1)
```

```
    source_nps_doc.append(tmp)
```

[15]:
```
# clean out stopwords
source_nps_doc_v2 = []

for file in source_nps_doc:
    file_temp = []
    for nps in file:
        if nps.lower() not in STOP_WORDS:
            file_temp.append(nps)
    source_nps_doc_v2.append(file_temp)
```

Performing term clustering

Training a feature representation for NPs

[17]:
```
# here we use word2vec algorithm to get the feature vectors of NPs
w2v_model = Word2Vec(source_nps_doc_v2, min_count=5) # Ignores all words with total frequency lower than 5.
w2v_feature = w2v_model.wv.vectors
```

Clustering NPs based on the feature representation

[18]:
```
kmeans = cluster.KMeans(n_clusters=10) # here we set the number of cluster is 10
kmeans.fit(w2v_feature)

labels = kmeans.labels_
```

Selecting the significant terms

[19]:
```
# define a function to get the top-n frequent terms from each cluster

def topN_NPs_per_clt(LABELS, W2V_NPS, W2V_NPS_CT, TOPN_NMB):
    topN_NPs_km = []

    for k in range(10):
        inx = np.where(LABELS == k)[0]

        w2v_NPs_t =  W2V_NPS[inx]
        w2v_NPs_ct_t = W2V_NPS_CT[inx]
        w2v_NPs_ct_dict_t = dict(zip(w2v_NPs_t, w2v_NPs_ct_t))

        topN_NPs_km.append(Counter(w2v_NPs_ct_dict_t).most_common()[:TOPN_NMB])

    return topN_NPs_km
```

[ ]:
```
# find the the top-10 frequent words in each cluster

w2v_NPs = np.array(list(w2v_model.wv.vocab.keys())) # NPs
w2v_NPs_ct = np.array([j.count for i, j in w2v_model.wv.vocab.items()]) # NPs' word count
topN_NPs_km = topN_NPs_per_clt(labels, w2v_NPs, w2v_NPs_ct, 10)

topN_NPs_km
```

Discovering NPs'relations from DBpedia

Testing whethter the NPs exist in DBpedia

[25]:
```
topN_NPs = [j[0] for i in topN_NPs_km for j in i]
topN_ct = [j[1] for i in topN_NPs_km for j in i]
topN_cl = [inx_i for inx_i, i in enumerate(topN_NPs_km) for j in i]

nps_exist_text_topN = []
nps_exist_link_topN = []
```

Use the *openSearch* API to match NPs to the existing string of DBpedia

[ ]:
```
S = requests.Session()
URL = "https://en.wikipedia.org/w/api.php"
for nps in topN_NPs:
    print(nps)
    PARAMS = {
        "action": "opensearch",
        "namespace": "0",
        "search": nps ,
        "limit": "5",
        "profile" : "fuzzy",
        "format": "json"}

    R = S.get(url=URL, params=PARAMS)
    DATA = R.json()

    if len(DATA[1]) == 0:
        nps_exist_text_topN.append([None])
```

```
        nps_exist_link_topN.append([None])
    else:
        nps_exist_text_topN.append(DATA[1])
        nps_exist_link_topN.append(DATA[3])
```

[27]:
```
# store the results into a dataframe
df_nps_exist = pd.DataFrame()

df_nps_exist['sourceNPs_counts'] = topN_ct
df_nps_exist['sourceNPs_cluster'] = topN_cl
df_nps_exist['sourceNPs'] = topN_NPs
df_nps_exist['WikiMatchedNPs'] = [i[0] for i in nps_exist_text_topN]  # extract the top one candidates
df_nps_exist['WikiMatchedLink'] = [i[0] for i in nps_exist_link_topN]   # extract the top one candidates
```

Discovering the knowledge if the NPs exist in DBpedia

[29]:
```
# format the recognized NPs and get the link ID of hypernym relation
np_query_li = ['_'.join(str(nps).split()) for nps in df_nps_exist.WikiMatchedNPs]
rel_query_li = ['<http://purl.org/linguistics/gold/hypernym>']

sparql = SPARQLWrapper("http://dbpedia.org/sparql")
```

[30]:
```
# query the OBJECT with given subject
NPs_AsSub_O = [None] * len(np_query_li)

for inx_np_query, np_query in enumerate(np_query_li):
    subj_qr = np_query
    obj_qr = np_query
    rel_qr = rel_query_li[0]

    NPs_AsSub_O_t = []

    try:
        # query the OBJECT with given subject
        sparql.setQuery("""
                PREFIX dbp: <http://dbpedia.org/resource/>
                PREFIX owl: <http://www.w3.org/2002/07/owl#>
                SELECT distinct ?y ?string where {
                   dbp:"""+subj_qr+" "+rel_qr+"""  ?y .
                   OPTIONAL {?y rdfs:label ?string . FILTER (lang(?string) = 'en') }}
        """)
        sparql.setReturnFormat(JSON)
        results = sparql.query().convert()

        for result in results["results"]["bindings"]:
            print('______')
            print(np_query)
            print(result["string"]["value"])
            NPs_AsSub_O_t.append(result["string"]["value"])

    except:
        # to exclude the bad formats errors
        print(' None exist')

    NPs_AsSub_O[inx_np_query] = NPs_AsSub_O_t
```

```
 None exist
 None exist
 None exist
 None exist
```

[ ]:
```
# query the SUBJECT with given object
NPs_AsObj_S = [None] * len(np_query_li)

for inx_np_query, np_query in enumerate(np_query_li):
    subj_qr = np_query
    obj_qr = np_query
    rel_qr = rel_query_li[0]

    NPs_AsObj_S_t = []


    try:
        # query the SUBJECT with given object
        sparql.setQuery("""
                PREFIX dbp: <http://dbpedia.org/resource/>
                PREFIX owl: <http://www.w3.org/2002/07/owl#>
                SELECT distinct ?y ?string  where {
                   ?y """+rel_qr+""" dbp:"""+obj_qr+""".
                   OPTIONAL {?y rdfs:label ?string . FILTER (lang(?string) = 'en') }}
        """)
        sparql.setReturnFormat(JSON)
        results = sparql.query().convert()

        for result in results["results"]["bindings"]:
            print('______')
            print(np_query)
            print(result["string"]["value"])
            NPs_AsObj_S_t.append(result["string"]["value"])

    except:
        # to exclude the bad formats errors
```

```
        print( "None exist " )

    NPs_AsObj_S[inx_np_query] = NPs_AsObj_S_t
```

Finished! Hope you enjoy this lab tutorial

[ ]: