NAGESH SINGH CHAUHAN · 1Y AGO · 8,869 VIEWS

Build knowledge graph using python

Python · Wiki Sentences

Run
4.9s

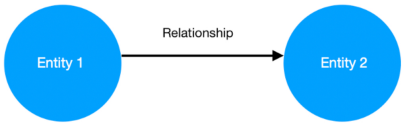
Version 1 of 1

pandas Matplotlib NLTK spaCy tqdm

What is a Knowledge Graph

A Knowledge Graph is a set of data points connected by relations that describe a domain, for instance, a business, an organization, or a field of study. It is a powerful way of representing data because Knowledge Graphs can be built automatically and can then be explored to reveal new insights about the domain.

The concept of Knowledge Graphs borrows from the Graph Theory. In this particular representation, we store data as:



Entity 1 and Entity 2 are called nodes and the Relationship is called an edge. Of course, in a real-world knowledge graph, there are lots of entities and relationships and there is more than one way to arrive at one entity starting from another.

Usually, these types of graphs is modeled with triples, which are sets of three items like (subject, verb, object), with the verb being the relationship between the subject and the object-for example (London, is_capital, England).

Table of Contents

What is a Knowledge Graph

In [1]:
```
import spacy
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/wiki-sentences/wiki_sentences_v2.csv
```

In [2]:
```
nlp = spacy.load('en_core_web_sm')

doc = nlp("The 22-year-old recently won ATP Challenger tournament.")

for tok in doc:
    print(tok.text, "...", tok.dep_)
```

```
The ... det
22-year ... compound
- ... punct
old ... nsubj
recently ... advmod
won ... ROOT
ATP ... compound
Challenger ... compound
tournament ... dobj
. ... punct
```

In [3]:
```
!pip install beautifulsoup4
```

```
Collecting beautifulsoup4
  Downloading beautifulsoup4-4.9.3-py3-none-any.whl (115 kB)
     |████████████████████████████████| 115 kB 420 kB/s eta 0:00:01
Collecting soupsieve>1.2
  Downloading soupsieve-2.2-py3-none-any.whl (33 kB)
Installing collected packages: soupsieve, beautifulsoup4
Successfully installed beautifulsoup4-4.9.3 soupsieve-2.2
WARNING: You are using pip version 21.0; however, version 21.0.1 is available.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip install --upgrade pi
p' command.
```

In [4]:
```
import re
import pandas as pd
import bs4
import requests
import spacy
from spacy import displacy
nlp = spacy.load('en_core_web_sm')

from spacy.matcher import Matcher
from spacy.tokens import Span

import networkx as nx

import matplotlib.pyplot as plt
from tqdm import tqdm

pd.set_option('display.max_colwidth', 200)
%matplotlib inline
```

In [5]:
```
# import wikipedia sentences
candidate_sentences = pd.read_csv("../input/wiki-sentences/wiki_sentences_v2.csv")
candidate_sentences.shape
```

Out[5]:
```
(4318, 1)
```

In [6]:
```
candidate_sentences
```

Out[6]:

	sentence
0	confused and frustrated, connie decides to leave on her own.
1	later, a woman's scream is heard in the distance.
2	christian is then paralyzed by an elder.
3	the temple is set on fire.
4	outside, the cult wails with him.
...	...
4313	confidencial also responded negatively, calling the film a barren drama, unsubtle and self-indulgent.
4314	and le parisien gave the film their highest five-star rating.
4315	the museum collection includes 37,000 film titles, 60,000 posters, 700,000 photographs and 20,000 books.
4316	its predecessor was the dutch historical film archive, founded in 1946.

4317	, 1920'sfilmstar greta garbo by alexander binder,

4318 rows × 1 columns

In [7]:
```
doc = nlp("confused and frustrated, connie decides to leave on her own.")

for tok in doc:
    print(tok.text, "...", tok.dep_)
```

```
confused ... advcl
and ... cc
frustrated ... conj
, ... punct
connie ... nsubj
decides ... ROOT
to ... aux
leave ... xcomp
on ... prep
her ... poss
own ... pobj
. ... punct
```

Entity Pairs Extraction

To build a knowledge graph, the most important things are the nodes and the edges between them.

In [8]:
```
def get_entities(sent):
    ## chunk 1
    ent1 = ""
    ent2 = ""

    prv_tok_dep = ""  # dependency tag of previous token in the sentence
    prv_tok_text = ""  # previous token in the sentence

    prefix = ""
    modifier = ""

    #############################################################

    for tok in nlp(sent):
        ## chunk 2
        # if token is a punctuation mark then move on to the next token
        if tok.dep_ != "punct":
            # check: token is a compound word or not
            if tok.dep_ == "compound":
                prefix = tok.text
                # if the previous word was also a 'compound' then add the current word to it
                if prv_tok_dep == "compound":
                    prefix = prv_tok_text + " " + tok.text

            # check: token is a modifier or not
            if tok.dep_.endswith("mod") == True:
                modifier = tok.text
                # if the previous word was also a 'compound' then add the current word to it
                if prv_tok_dep == "compound":
                    modifier = prv_tok_text + " " + tok.text

            ## chunk 3
            if tok.dep_.find("subj") == True:
                ent1 = modifier + " " + prefix + " " + tok.text
                prefix = ""
                modifier = ""
                prv_tok_dep = ""
                prv_tok_text = ""

                ## chunk 4
            if tok.dep_.find("obj") == True:
                ent2 = modifier + " " + prefix + " " + tok.text

            ## chunk 5
            # update variables
            prv_tok_dep = tok.dep_
            prv_tok_text = tok.text
    #############################################################

    return [ent1.strip(), ent2.strip()]
```

In [9]:
```
get_entities("the film had 200 patents")
```

Out[9]:
```
['film', '200  patents']
```

In [10]:
```
entity_pairs = []

for i in tqdm(candidate_sentences["sentence"]):
    entity_pairs.append(get_entities(i))
```

```
100%|████████| 4318/4318 [00:40<00:00, 105.72it/s]
```

In [11]:
```
entity_pairs[10:20]
```

Out[11]:
```
[['we', 'tests'],
 ['m global', 'international sales rights'],
 ['canadian musician robbie robertson', 'soundtrack'],
 ['it', 'original music tracks'],
 ['it', 'reviewed  franchise'],
 ['she', 'accidentally  mystique'],
 ['military  forces', 'arrest'],
 ['train', 'vuk'],
 ['', 'telepath selene gallio'],
 ['singer', 'sequel']]
```

Relation / Predicate Extraction

In [12]:
```
def get_relation(sent):

    doc = nlp(sent)

    # Matcher class object
    matcher = Matcher(nlp.vocab)

    #define the pattern
    pattern = [{'DEP':'ROOT'},
            {'DEP':'prep','OP':"?"},
            {'DEP':'agent','OP':"?"},
            {'POS':'ADJ','OP':"?"}]

    matcher.add("matching_1", None, pattern)

    matches = matcher(doc)
    k = len(matches) - 1

    span = doc[matches[k][1]:matches[k][2]]

    return(span.text)
```

In [13]:
```
get_relation("John completed the task")
```

Out[13]:
```
'completed'
```

In [14]:
```
relations = [get_relation(i) for i in tqdm(candidate_sentences['sentence'])]
```

```
100%|████████| 4318/4318 [00:38<00:00, 112.10it/s]
```

In [15]:
```
pd.Series(relations).value_counts()[:50]
```

Out[15]:
```
is                372
was               302
released on        88
are                74
include            72
were               67
released           42
's                 39
composed by        35
became             31
have               31
has                30
become             28
released in        27
included           25
called             21
had                21
made               20
considered         20
been               19
produced           19
used               18
scheduled          17
be                 16
received           15
written by         15
hired              14
produced by        14
wrote              14
stars              14
introduced in      13
went               13
directed by        12
wanted             12
began in           11
began              11
won                11
sold               10
set                10
gave               10
features           10
used in             9
shot in             9
includes            9
produced in         9
reported            9
cast as             9
opened              9
said                9
revealed            8
dtype: int64
```

Build a Knowledge Graph

In [16]:
```
# extract subject
source = [i[0] for i in entity_pairs]

# extract object
target = [i[1] for i in entity_pairs]

kg_df = pd.DataFrame({'source':source, 'target':target, 'edge':relations})
kg_df
```

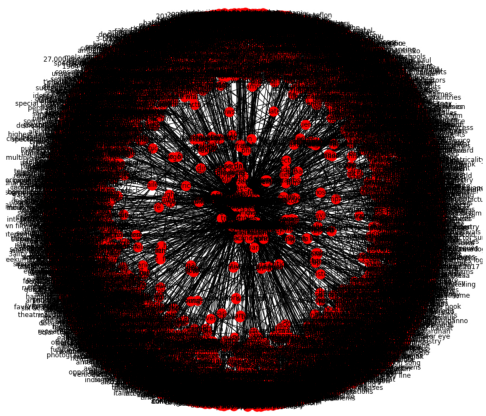
Out[16]:

	source	target	edge
0	connie	own	decides
1	later woman	distance	heard in
2	christian	then elder	paralyzed by
3	temple	fire	set on
4	outside cult	him	wails with
...	...	...	...
4313	confidencial	negatively film	responded
4314	le parisien	five star rating	gave
4315	museum collection	37,000 film titles	includes
4316	predecessor	archive 1946	was
4317		1920'sfilmstar alexander binder	garbo by

4318 rows × 3 columns

In [17]:
```
# create a directed-graph from a dataframe
G=nx.from_pandas_edgelist(kg_df, "source", "target",
                          edge_attr=True, create_using=nx.MultiDiGraph())
```

In [18]:
```
plt.figure(figsize=(12,12))

pos = nx.spring_layout(G)
nx.draw(G, with_labels=True, node_color='red', edge_cmap=plt.cm.Blues, pos = pos)
plt.show()
```

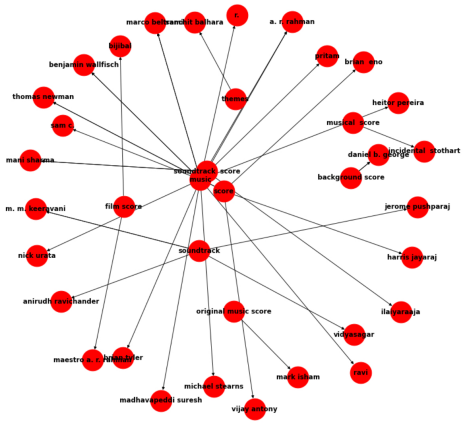


Well, this is not exactly what we were hoping for (still looks quite a sight though!).

It turns out that we have created a graph with all the relations that we had. It becomes really hard to visualize a graph with these many relations or predicates.

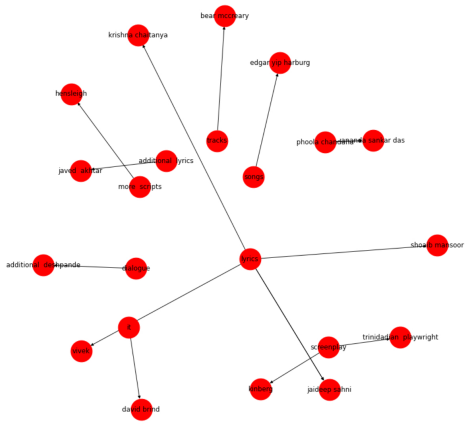
So, it's advisable to use only a few important relations to visualize a graph. I will take one relation at a time. Let's start with the relation "composed by":

In [19]:
```
G=nx.from_pandas_edgelist(kg_df[kg_df['edge']=="composed by"], "source", "target",
                          edge_attr=True, create_using=nx.MultiDiGraph())

plt.figure(figsize=(12,12))
pos = nx.spring_layout(G, k = 0.5) # k regulates the distance between nodes
nx.draw(G, with_labels=True, node_color='red', node_size=1500, edge_cmap=plt.cm.Blues, pos = po
s, font_weight='bold')
plt.show()
```



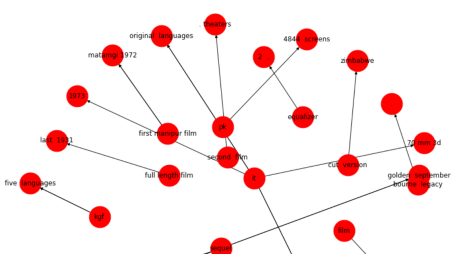
Since writing is an important role in any movie, I would like to visualize the graph for the "written by" relation:

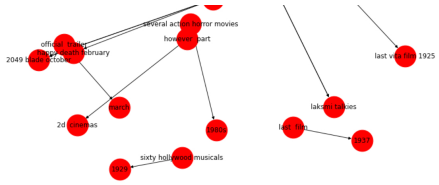
In [20]:
```
G=nx.from_pandas_edgelist(kg_df[kg_df['edge']=="written by"], "source", "target",
                          edge_attr=True, create_using=nx.MultiDiGraph())

plt.figure(figsize=(12,12))
pos = nx.spring_layout(G, k = 0.5)
nx.draw(G, with_labels=True, node_color='red', node_size=1500, edge_cmap=plt.cm.Blues, pos = po
s)
plt.show()
```



Let's see the knowledge graph of another important predicate, i.e., the "released in":

In [22]:
```
G=nx.from_pandas_edgelist(kg_df[kg_df['edge']=="released in"], "source", "target",
                          edge_attr=True, create_using=nx.MultiDiGraph())

plt.figure(figsize=(12,12))
pos = nx.spring_layout(G, k = 0.5)
nx.draw(G, with_labels=True, node_color='red', node_size=1500, edge_cmap=plt.cm.Blues, pos = po
s)
plt.show()
```





In [ ]:

License

Continue exploring

Data
1 input and 0 output →

Logs
4.9 second run - successful →

Comments
8 comments →