Coreference resolution of the <PERSON> entity

GOURAV SINHA · Following
6 min read · Jun 5

The world of AI is enthusiastically embracing large language models (LLMs) for their striking advantages, which can simplify or even eliminate the need for text preprocessing. I must admit they work in many use cases, but there are still cases where conventional models are better. You may refer to this paper for valid comparisons https://arxiv.org/pdf/2304.13712v2.pdf. You should also be aware that some LLMs that you access via APIs charge you based on the number of tokens used. To perform tasks such as entity extraction, sentiment analysis, and relationship extraction, you must preserve the context in a large corpus. There are models available, such as MPT-7Bs, which have token limits of up to 84k tokens. However, when it comes to tasks such as entity or relationship extraction, you will not only need to fine-tune the model, but you will also need to perform some text preprocessing to make model training and inference easier. In my next blog I will discuss more on LLM finetuning and will also help my readers to perhaps build and train their own LLMs 😊(I have achieved it and will help you too).
Let's get back to the topic of our discussion. Coreference resolution is a preprocessing technique whose importance can never be understated. Unfortunately, there are very few articles available online that help you build an entire coreference resolution pipeline from start to finish.

I will be using spacy experimental to create coreference clusters.
So let's start:

You may use latest version of spacy. In this particular code I have used spacy==3.5.2.

Installations and downloads:

Note : Do not install spacy after you have installed spacy-experimental. Do it

before installing the spacy-experimental. Also download en_core_web_trf after you have installed the below packages.

```
!pip install spacy-experimental
!pip install https://github.com/explosion/spacy-experimental/releases/download/v
```

```
!spacy download en_core_web_trf
```

Imports:

```
import spacy
import spacy_experimental
import re
import pandas as pd
```

The entire coreference resolution pipeline in my code will be inside the `coref_resolution` class. I will be explaining each and every class method accordingly below:

```
def get_coref_clusters(self,):
    """This method produces coref clusters"""
    self.nlp = spacy.load("en_core_web_trf")
    nlp_coref = spacy.load("en_coreference_web_trf")

    nlp_coref.replace_listeners("transformer", "coref", ["model.tok2vec"])
    nlp_coref.replace_listeners("transformer", "span_resolver", ["model.tok2vec"

    self.nlp.add_pipe("coref", source=nlp_coref)
```

```
        self.nlp.add_pipe("span_resolver", source=nlp_coref)

        self.doc = self.nlp(self.text)
        self.tokens = [str(token) for token in self.doc]
        coref_clusters = {key : val for key , val in self.doc.spans.items() if re.ma

        return coref_clusters
```

The function first loads two SpaCy models: `en_core_web_trf` and `en_coreference_web_trf`. The `en_core_web_trf` model is a general-purpose language model, while the `en_coreference_web_trf` model is a specialized model for coreference resolution.

The function then uses the `replace_listeners()` method to add two listeners to the `en_coreference_web_trf` model. These listeners will be called whenever the model encounters a coreference mention or a span.

The function then adds the `coref` and `span_resolver` pipes to the `en_core_web_trf` model. These pipes will be used to perform coreference resolution and resolve spans, respectively.

The function then uses the `doc` method to create a document object from the text. The `tokens` method is then used to extract the tokens from the document object.

The `re.match()` method is then used to find all of the cluster names in the document object that match the regex `r"coref_clusters_*"`. These clusters are then grouped together into a dictionary, with the keys being the names of the clusters and the values being list of the <PERSON> entity and all its references. example :

```
text = """John took music class even though he resented it. His interest was mor
obj = coref_resolution(text)
coref_clusters = obj.get_coref_clusters()
print(coref_clusters)
```

```
###Output:###
{'coref_clusters_1': [John took, he resented, His interest],
'coref_clusters_2': [music class even, it.]}
```

The function finally returns the dictionary of coreference clusters, as mentioned above. In the above clusters, the first element of each cluster is the actual person, while the rest of the elements are its references. For example, in the `coref_clusters_1` cluster, John is being referred to by "he" and "His" from the subsequent list elements.

Now, lets gather start and end token positions of each and every cluster elements from the coref_clusters :

```
def find_span_start_end(self,coref_clusters):

    cluster_w_spans = {}
    for cluster in coref_clusters:
      cluster_w_spans[cluster] = [(span.start, span.end, span.text) for span in

    return cluster_w_spans
```

```
cluster_w_spans = obj.find_span_start_end(coref_clusters)
print(cluster_w_spans)

## Output:
```

```
## {'coref_clusters_1': [(0, 2, 'John took'), (6, 8, 'he resented'), (10, 12, 'H
#'coref_clusters_2': [(2, 5, 'music class even'), (8, 10, 'it.')]}
```

Next function might seem a bit complex but the purpose of this function is to find <PERSON> entity and its corresponding references.

```
def find_person_start_end(self, coref_clusters,cluster_w_spans):
    # nlp = spacy.load("en_core_web_trf")
    coref_clusters_with_name_spans = {}
    for key, val in coref_clusters.items():
      temp = [0 for i in range(len(val))]
      person_flag = False
      for idx, text in enumerate(val):
        doc = self.nlp(str(text))
        for word in doc.ents:
          # find the absolute token position of PERSON entity
          if word.label_ == 'PERSON':
            temp[idx] = (word.start, word.end, word.text)
            person_flag = True
        for token in doc:
          # find the absolute token position of the pronouns and references
          if token.pos_ == 'PRON':
            temp[idx] = (token.i,token.i+1,token)
      if len(temp) > 0:
        # replace the absolute token positions with the relative token positions
        if person_flag:
          orig = cluster_w_spans[key]
          for idx, tup in enumerate(orig):
            if isinstance(tup, tuple) and isinstance(temp[idx], tuple):
              orig_start, orig_end, text = tup
              offset_start, offset_end, _ = temp[idx]
              orig_start += offset_start
              orig_end = orig_start + (offset_end - offset_start)
              orig[idx] = (orig_start, orig_end, text)
          coref_clusters_with_name_spans[key] = orig

    return coref_clusters_with_name_spans
```

In a nutshell this function calculates the relative token positions of the PERSON entity tokens and the corresponding references from the entire corpus.
You could also get coref_head_clusters directly but getting token position in that case is lot more difficult.

```
coref_clusters_with_name_spans = obj.find_person_start_end(coref_clusters,cluste
print(coref_clusters_with_name_spans)

## output :
## {'coref_clusters_1': [(0, 1, 'John took'), (6, 7, 'he resented'), (10, 11, 'H
```

Now after we get positions of entity and reference POS tags we replace the POS tags with names. The code also takes care apostrophe rules.

```
def replace_refs_w_names(self,coref_clusters_with_name_spans):
    tokens = self.tokens
    special_tokens = ["my","his","her","mine"]
    for key, val in coref_clusters_with_name_spans.items():
      if len(val) > 0 and isinstance(val, list):
        head = val[0]
        head_start, head_end, _ = head
        head_name = " ".join(tokens[head_start:head_end])
        for i in range(1,len(val)):
          coref_token_start, coref_token_end, _ = val[i]
          count = 0
          for j in range(coref_token_start, coref_token_end):
            if tokens[j].upper() == "I":
                count += 1
                continue
            if count == 0:
              if tokens[j].lower() in special_tokens:
                if head_name[-1].lower() == "s":
                  tokens[j] = str(head_name)+"'"
                else:
                  tokens[j] = str(head_name)+"'s"
```

```
                else:
                  tokens[j] = head_name
              else:
                tokens[j] = ""
              count += 1

    return tokens
```

```
tokens = obj.replace_refs_w_names(coref_clusters_with_name_spans)
print(" ".join(tokens))

## Output : John took music class even though John resented it . John's interest
```

Now lets try with full names:

```
text = """James Morrison and Michael jackson met at a bar in paradise. Jim inqui
obj = coref_resolution(text)
coref_clusters = obj.get_coref_clusters()
# print(coref_clusters)
cluster_w_spans = obj.find_span_start_end(coref_clusters)
coref_clusters_with_name_spans = obj.find_person_start_end(coref_clusters,cluste
tokens = obj.replace_refs_w_names(coref_clusters_with_name_spans)
print(" ".join(tokens))

## Output : James Morrison and Michael jackson met at a bar in paradise . James
```

Wrapping up all the code in a python Class:

```
class coref_resolution:
  def __init__(self,text):
    self.text = text

  def get_coref_clusters(self,):
```

```
        self.nlp = spacy.load("en_core_web_trf")
        nlp_coref = spacy.load("en_coreference_web_trf")

        nlp_coref.replace_listeners("transformer", "coref", ["model.tok2vec"])
        nlp_coref.replace_listeners("transformer", "span_resolver", ["model.tok2vec"

        self.nlp.add_pipe("coref", source=nlp_coref)
        self.nlp.add_pipe("span_resolver", source=nlp_coref)

        self.doc = self.nlp(self.text)
        self.tokens = [str(token) for token in self.doc]
        coref_clusters = {key : val for key , val in self.doc.spans.items() if re.ma

        return coref_clusters

    def find_span_start_end(self,coref_clusters):

        cluster_w_spans = {}
        for cluster in coref_clusters:
            cluster_w_spans[cluster] = [(span.start, span.end, span.text) for span in

        return cluster_w_spans

    def find_person_start_end(self, coref_clusters,cluster_w_spans):
        # nlp = spacy.load("en_core_web_trf")
        coref_clusters_with_name_spans = {}
        for key, val in coref_clusters.items():
            temp = [0 for i in range(len(val))]
            person_flag = False
            for idx, text in enumerate(val):
                doc = self.nlp(str(text))
                for word in doc.ents:
                    # find the absolute token position of PERSON entity
                    if word.label_ == 'PERSON':
                        temp[idx] = (word.start, word.end, word.text)
                        person_flag = True
                for token in doc:
                    # find the absolute token position of the pronouns and references
                    if token.pos_ == 'PRON':
                        temp[idx] = (token.i,token.i+1,token)
            if len(temp) > 0:
                # replace the absolute token positions with the relative token positions
                if person_flag:
                    orig = cluster_w_spans[key]
                    for idx, tup in enumerate(orig):
                        if isinstance(tup, tuple) and isinstance(temp[idx], tuple):
                            orig_start, orig_end, text = tup
                            offset_start, offset_end, _ = temp[idx]
                            orig_start += offset_start
                            orig_end = orig_start + (offset_end - offset_start)
```

```
                orig[idx] = (orig_start, orig_end, text)
            coref_clusters_with_name_spans[key] = orig

    return coref_clusters_with_name_spans

  def replace_refs_w_names(self,coref_clusters_with_name_spans):
    tokens = self.tokens
    special_tokens = ["my","his","her","mine"]
    for key, val in coref_clusters_with_name_spans.items():
      if len(val) > 0 and isinstance(val, list):
        head = val[0]
        head_start, head_end, _ = head
        head_name = " ".join(tokens[head_start:head_end])
        for i in range(1,len(val)):
          coref_token_start, coref_token_end, _ = val[i]
          count = 0
          for j in range(coref_token_start, coref_token_end):
            if tokens[j].upper() == "I":
                count += 1
                continue
            if count == 0:
              if tokens[j].lower() in special_tokens:
                if head_name[-1].lower() == "s":
                  tokens[j] = str(head_name)+"'"
                else:
                  tokens[j] = str(head_name)+"'s"
              else:
                tokens[j] = head_name
            else:
              tokens[j] = ""
            count += 1

    return tokens

  def main(self,):

    coref_clusters = self.get_coref_clusters()
    coref_w_spans = self.find_span_start_end(coref_clusters)
    coref_clusters_with_name_spans = self.find_person_start_end(coref_clusters,c
    tokens = self.replace_refs_w_names(coref_clusters_with_name_spans)

    return " ".join(tokens)
```

Thanks for reading till this point. Hope you enjoyed this tutorial. I will publish more blogs on LLMs and finetuning.

You could follow me on linkedin : https://www.linkedin.com/in/gourav-sinha-18b9a026/

My github : https://github.com/gouravsinha1405/

Coreference Resolution | Naturallanguageprocessing | Machine Learning | Deep Learning | Large Language Models

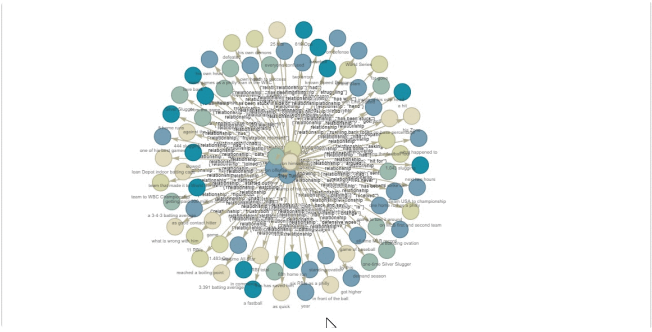
Written by GOURAV SINHA

2 Followers

Data Scientist/NLP engineer

Following

Recommended from Medium



Wenqi Glantz in Better Programming

7 Query Strategies for Navigating Knowledge Graphs With...

Exploring NebulaGraph RAG Pipeline with the Philadelphia Phillies

17 min read · 4 days ago

501 4

Pierre-Sylvain AUGEREAU

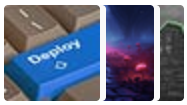
The power of named Entity Recognition (NER) with LLM

How does Large Language Model level up Named Entity Recognition?

3 min read · May 14

6

Lists



Predictive Modeling w/ Python

20 stories · 452 saves

Natural Language Processing

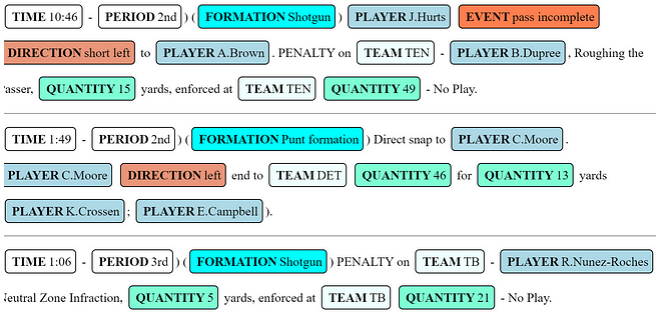
669 stories · 283 saves

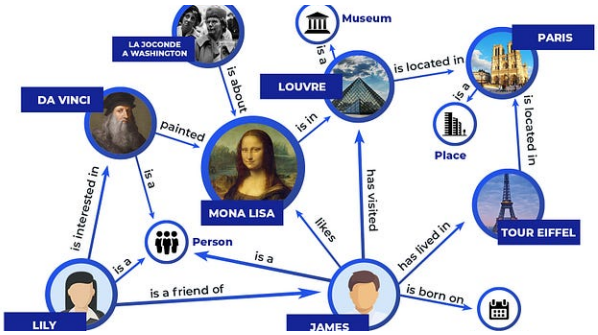
Practical Guides to Machine Learning

10 stories · 519 saves

AI Regulation

6 stories · 138 saves





dp

Alla Chepurova in DeepPavlov

Building Custom Named-Entity Recognition (NER) Models

Complete walk-through where we start with a dataset, quickly annotate our dataset...

6 min read · Apr 29

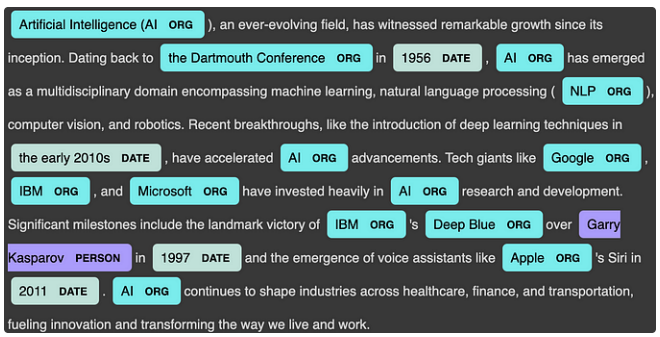
25 2

Improving Knowledge Graph Completion with Generative LM...

Combining both internal LM knowledge and external data from KG

13 min read · Sep 5

36



Angelica Lo Duca in Towards Data Science

Sahithi Reddy

How to Create a Custom NER in Spacy 3.5

A quick tutorial on extracting custom entities from a text

✨ · 5 min read · Apr 25

42 2

Named Entity Recognition using Python and Spacy.

NER, short for Named Entity Recognition, is an NLP method that detects and categorizes...

4 min read · Jun 11

24

See more recommendations