P3 Avistamiento de aves

- autor: Miquel Antoni Llambías Cabot
- repository: github

Obtener datos (scraping)

Primero vamos a obtener los datos. Para ello nos visitaremos las web de avistamientos de aves shorebirder, trevorsbirding y dantallmansbirdblog.

Durante la visita a la web y haciendo uso del inspector (F12) podemos ver que las descripciones que necesitamos se encuentran en los tag de párrafo (entre * TEXTO *). Sabiendo eso vamos a crear funciones de utilidad que se encargarán de descargar el contenido de la web y extraer el texto.

Las descargas las realizaremos en `data/raw` mientras que en `data/posts` guardaremos los textos encontrados.

** `dantallmansbirdblog` tiene una estructura ligeramente diferente (entre * TEXTO *), a lo que tendremos que modificar la función `get_texts` (a continuación) para obtener sus textos.

```
# I/O utils
import os
from os.path import exists
import re
import wget
import tqdm
import json

# auxiliar regex
remove_parentesis_text = re.compile(r'\(.*\)')
sentence_endings = re.compile(r'\.|\*')

# auxiliar functions
def maybe_mkdir(path):
  try:
    if not exists(path):
      os.mkdir(path)
  except OSError as error:
    print(error)

def download(url, out_label):
  filepath = f"{data_raw_path}/{out_label}"
  if exists(filepath):
    os.remove(filepath)
  return wget.download(url, out=filepath)

def get_texts(filename):
  file = open(filename, 'r')
  text = file.read()
  file.close()
```

```
    # get texts
    get_p = re.compile(r'<p>((.|\n)*?)</p>')
    texts = get_p.findall(text)

    # remove styling and inner tags
    remove_tags = re.compile(r'(<.*?>)|\\n| +(?= )|\\|\&.+?\;')
    return map(lambda text: re.sub(remove_tags, "", str(text[0]).lower()), texts)

  def write(path, filename, data):
    filepath = f"{path}/{filename}.txt"
    file = open(filepath, "a", encoding="utf-8")
    for item in data:
      file.write(str(item)+"\n")
    file.close()
    return filepath

  def save_json(path, filename, dict):
    filepath = f"{path}/{filename}.json"
    # create json object from dictionary
    parsed_json = json.dumps(dict)
    f = open(filepath,"w")
    # write json object to file
    f.write(parsed_json)
    # close file
    f.close()
    return filepath

  # constants
  data_posts_cache = "../data/cache" # guardar resultados de queries a sparql
  data_raw_path = "../data/raw" # descargas
  data_posts_path = "../data/posts" # guardar los textos de los post scrapeados
  data_results_path = "../data/results" # guardar los resultados de las diferentes
  pruebas

  # build directory structure
  maybe_mkdir("../models")
  maybe_mkdir("../models/entity_ruler")
  maybe_mkdir("../data")
  maybe_mkdir("../owl")
  maybe_mkdir(data_raw_path)
  maybe_mkdir(data_posts_path)
  maybe_mkdir(data_results_path)
```

A continuación haciendo uso de las funciones anteriores scrapeamos la home de `shorebirder`.

```
  # scrap shorebirder.com
  reviews_filename = "reviews"
  shorebirder_filename = "shorebirder_home.html"
  shorebirder_home = download("https://www.shorebirder.com/", shorebirder_filename)
  posts = get_texts(shorebirder_home)
  shorebirder_posts_file = write(data_posts_path, reviews_filename, posts)
```

```
open(shorebirder_posts_file, "r", encoding="utf-8").readlines()[0]
```

```
"my late march solo visit to norway is in the books and was about as much fun as
i've had in a while. the middle few days of the trip were spent birding around
varanger, bookended by more touristy time intromsã¸and oslo. at some point in the
coming months there will be a full trip report here plus a very detailed
cloudbirders submission. in the meantime, here is some proof that i actually
went.\n"
```

Lo mismo para `trevorsbirding`.

```
# scrap trevorsbirding.com
trevorsbirding_filename = "trevorsbirding_home.html"
trevorsbirding_home = download("https://www.trevorsbirding.com/",
trevorsbirding_filename)
posts = get_texts(trevorsbirding_home)
trevorsbirding_posts_file = write(data_posts_path, reviews_filename, posts)

open(trevorsbirding_posts_file, "r").readlines()[2]
```

```
'i kid. new haven gets a bad rap, and it really shouldn\'t. or is it "bad rep??" i
just don\'t know.\n'
```

No se han obtenido las descripciones de `dantallmansbirdblog` dado que con los dos primeros ya cubrimos el "mínimo" frases de posibles avistamientos.

Intento 1: Usar spacy sin modificar

```
# Cargar dependencias para el nlp
import spacy
from spacy import displacy
nlp = spacy.load("en_core_web_lg")
```

```
# Prueba
for text in open(shorebirder_posts_file, "r").readlines()[3:5]:
  train = nlp(text)
  displacy.render(train, jupyter=True, style="ent")
```

new haven **GPE** : come for the pizza, stay for the crime!

last week DATE i was out messing with my brand new camera body, the canon r5 WORK_OF_ART , which at the time i had paired with my trusty ol' 400mm QUANTITY f5.6. i was walking back to the car at fort hale park FAC when i heard a "thwack!" right next to me followed by the most awful gull scream you've ever heard. i looked down to my left and a ring-billed gull was flailing a bloody, broken right wing.

looking up i eventually located the culprit, a hefty adult (presumed) female peregrine falcon.

Vemos que es capaz de identificar diferentes entidades dentro de las frases, pero no pájaros.

Intento 2: Usando sparql query para encontrar aves

Idea: Usando el tokenizer de spacy como tokenizer trocear las frases. A partir de los token etiquetados como nombre (NOUN) lanzamos una petición a la dbpedia. Ya el resultado de la dbpedia nos dirá si existe y cual es su etiqueta / url.

```
# SparQL class extension
# Prefixes and Class based from https://github.com/ejrav/pydbpedia
from SPARQLWrapper import SPARQLWrapper, JSON

class SparqlEndpoint(object):

    def __init__(self, endpoint, prefixes={}):
        self.sparql = SPARQLWrapper(endpoint)
        self.prefixes = {
            "dbo": "http://dbpedia.org/ontology/",
            "owl": "http://www.w3.org/2002/07/owl#",
            "xsd": "http://www.w3.org/2001/XMLSchema#",
            "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
            "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
            "foaf": "http://xmlns.com/foaf/0.1/",
            "dc": "http://purl.org/dc/elements/1.1/",
            "dbpedia2": "http://dbpedia.org/property/",
            "dbpedia": "http://dbpedia.org/",
            "skos": "http://www.w3.org/2004/02/skos/core#",
            "foaf": "http://xmlns.com/foaf/0.1/",
            "yago": "http://dbpedia.org/class/yago/",
            }
        self.prefixes.update(prefixes)
        self.sparql.setReturnFormat(JSON)

    def query(self, q):
        lines = ["PREFIX %s: <%s>" % (k, r) for k, r in self.prefixes.items()]
        lines.extend(q.split("\n"))
        query = "\n".join(lines)
        self.sparql.setQuery(query)
        results = self.sparql.query().convert()
        return results["results"]["bindings"]
```

```
class DBpediaEndpoint(SparqlEndpoint):
    def __init__(self, endpoint, prefixes = {}):
        super(DBpediaEndpoint, self).__init__(endpoint, prefixes)

s = DBpediaEndpoint(endpoint = "http://dbpedia.org/sparql")
```

```
# Función para buscar una ave dado su nombre
def search_bird_dbpedia(token):
  return s.query('''
    SELECT *
    WHERE {
      ?bird a dbo:Bird ;
            rdfs:label ?name ;
            dbo:abstract ?comment .

      filter (!isLiteral(?name) ||
              langmatches(lang(?name), "en")) .

      filter (!isLiteral(?comment) ||
              langmatches(lang(?comment), "en")) .

      filter (CONTAINS(LCASE(STR(?name)), "{token}")) .
    }
    limit 5
  '''.replace("{token}", token))

search_bird_dbpedia("falcon")
```

```
# Prueba
nlp = spacy.load("en_core_web_lg")
maybe_matches = {}
for text in open(shorebirder_posts_file, "r").readlines()[4:5]:
  doc = nlp(text)
  for chunk in doc.noun_chunks:
    for token in chunk:
      if token.pos_ == 'NOUN':
        results = search_bird_dbpedia(token.lemma_)
        if len(results) > 0:
          maybe_matches[token] = results
          print(token)
```

```
body
time
ol'
car
gull
ring
```

```
gull
wing
peregrine
falcon
```

Vale, funciona? Lo que es muy lento y estamos machacando la dbpedia a queries.

Intento 3: Cachear / indexar la dbpedia

Del intento anterior vamos a coger los resultados de todos los pájaros y lo convertiremos en un diccionario para que nos sea más fácil buscar y solo haremos n queries a la dbpedia. Por supuesto, esta estrategia es solo factible si el conjunto es finito. Como es nuestro caso, va haber n especies de pájaros, pero no va a estar creciendo dia a dia.

Estrategia

- Obtener lista de todos los nombres de pájaros.
- Con spacy analizaremos la entrada del avistamiento y obtenemos los `noun chunk` y `ents`.
  - Para obtener los chunk necesitamos las pipelines de `tok2vec`, `tagger`, `parser` y `attribute_ruler`.
  - Para obtener las entidades necesitamos `ner`.
- Con cada `chunk` usando fuzzy-search en la lista de nombres de pájaros para encontrar aquellos chunk que parezcan nombres de pájaros.
- Con cada `ent` nos aportará datos sobre el contexto: ubicación, fechas u otros. Este ent se asociará al pájaro si el ent pertenece al chunk del pájaro y en su defecto al avistamiento.

```
# Obtener todos los pájaros con nombre, url y descripción
def get_all_birds_sparql(batch_size):
  found_all = False
  limit = batch_size
  offset = 0
  query = """
    SELECT DISTINCT *
    WHERE {
      ?bird a dbo:Bird ;
            rdfs:label ?name ;
            dbo:abstract ?comment .

      filter (!isLiteral(?name) ||
              langmatches(lang(?name), "en")) .

      filter (!isLiteral(?comment) ||
              langmatches(lang(?comment), "en")) .

    }
    limit {limit_value}
    offset {offset_value}
  """
  result = []
```

```
  while not found_all:
    loop_result = s.query(query.replace("{limit_value}", str(limit)).replace("
{offset_value}", str(offset)))
    if len(loop_result) < limit:
      found_all = True
    else:
      offset = limit
      limit += batch_size
    result += loop_result
  return result

birds_sparql = get_all_birds_sparql(5000)

write("../data", "birds", birds_sparql)

print(f"Hemos obtenido los nombres de {len(birds_sparql)} pájaros")
for d in birds_sparql[0:5]:
  print(d['name']['value'])
```

```
Hemos obtenido los nombres de 10369 pájaros
Actenoides
African goshawk
African pitta
African red-eyed bulbul
Alcedo
```

Convertimos los datos en crudo a un diccionario

```
# sparql a diccionario
remove_parentesis_text = re.compile(r'\(|\)')

birds = {}
for bird in birds_sparql:
  bird_name_lower = bird["name"]["value"].lower()
  key = re.sub(remove_parentesis_text, "", str(bird_name_lower))
  if key in birds.keys():
    print(f"bird {key} is duplicated.", birds[key]["name"], bird["name"]["value"])
  birds[key] = {
    "name": bird["name"]["value"],
    "url": bird["bird"]["value"],
    "description": bird["comment"]["value"],
  }
bird_keys = birds.keys() # buscaremos por las key
assert len(birds_sparql) == len(bird_keys) # aseguramos que no hayamos perdido
ninguna key
```

```
birds["goliath mangalia"] # bueno... la dbpedia tampoco es perfecta
```

```
{'name': 'Goliath (Mangalia)',
 'url': 'http://dbpedia.org/resource/Goliath_(Mangalia)',
 'description': 'Goliath is the name of a crane that is currently located at the
Mangalia shipyard in Mangalia, Romania. Formerly, it was part of the Fore River
Shipyard in Quincy, Massachusetts.'}
```

Preparamos el `nlp`. Usamos el preset large (`en_core_web_lg`) porque nos proporciona más precisión en el reconocimiento de entidades. Este reconocimiento de entidades lo usaremos para registrar elementos del contexto y completar los individuos de nuestra ontología

```
# preparar nlp
nlp = spacy.load("en_core_web_lg", disable=['lemmatizer'])
print(nlp.pipe_names)

doc = nlp("Black-billed flycatcher")
displacy.render(doc, jupyter=True)
```

```
['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'ner']
```

Black-

ADJ

billed VERB flycatcher NOUN npadvmod amod

Podemos ver que el `tokenizer` de spacy nos separa las palabras compuestas con guion.

Para solucionarlo vamos a modificar el tokenizer para que no separe las palabras con guion.

```
# hacer que el tokenizer no separe palabras con guion
# https://stackoverflow.com/questions/59993683/how-can-i-get-spacy-to-stop-
splitting-both-hyphenated-numbers-and-words-into-sep

from spacy.tokenizer import Tokenizer
```

```
from spacy.util import compile_infix_regex

def custom_tokenizer(nlp):
    inf = list(nlp.Defaults.infixes)               # Default infixes
    inf.remove(r"(?<=[0-9])[+\-\*^](?=[0-9-])")    # Remove the generic op between
numbers or between a number and a -
    inf = tuple(inf)                               # Convert inf to tuple
    infixes = inf + tuple([r"(?<=[0-9])[+*^](?=[0-9-])", r"(?<=[0-9])-(?=-)"])  #
Add the removed rule after subtracting (?<=[0-9])-(?=[0-9]) pattern
    infixes = [x for x in infixes if '-|–|—|--|---|——|~' not in x] # Remove -
between letters rule
    infix_re = compile_infix_regex(infixes)

    return Tokenizer(nlp.vocab, prefix_search=nlp.tokenizer.prefix_search,
                                suffix_search=nlp.tokenizer.suffix_search,
                                infix_finditer=infix_re.finditer,
                                token_match=nlp.tokenizer.token_match,
                                rules=nlp.Defaults.tokenizer_exceptions)
```

```
# Test
nlp.tokenizer = custom_tokenizer(nlp)

doc = nlp("Black-billed flycatcher")
displacy.render(doc, jupyter=True)
```

Black-billed

ADJ

flycatcher NOUN amod

Genial! A demás conseguimos que `Black-billed` se detecte como adjetivo y no como adjetivo + verbo.

A continuación, creamos las funciones necesarias para detectar los pájaros y una demo/test del funcionamiento

```
# Busqueda de pájaros por fuzzysearch
from fuzzywuzzy import fuzz
def search_bird_dict(chunk):
  best_match = (False, None, 0)
```

```
  for key in bird_keys:
    score = fuzz.ratio(key, chunk)
    if score > best_match[-1]:
      # print(f"\tchunk: '{chunk}' compare '{key}' score: '{score}'")
      best_match = (score > 80, key, score)
  return best_match

def get_nouns(chunk):
  only_nouns = []
  for token in chunk:
    if token.pos_ == "NOUN" or token.pos_ == "ADJ":
      only_nouns.append(token.lower_)

  if len(only_nouns) > 0:
    return " ".join(only_nouns)
  return None

maybe_matches = []
for text in open(shorebirder_posts_file, "r").readlines()[0:5]:
  doc = nlp(text)
  for chunk in [get_nouns(chunk) for chunk in doc.noun_chunks]:
  # for chunk in doc.noun_chunks: # aplicar sugerencia a ver si es más rápido así
-- 1.0s +/- 0.1s más lento que el anterior -- no y perdemos precisión
  #   chunk = str(chunk)
    if chunk != None:
      (found, bird_key, score) = search_bird_dict(chunk)
      if found:
        maybe_matches.append(bird_key)
        print(f"Found '{bird_key}' in '{chunk}' with score of {score}")

# Test result
print(maybe_matches)
```

```
Found 'ring-billed gull' in 'ring-billed gull' with score of 100
Found 'peregrine falcon' in 'female peregrine falcon' with score of 82
['ring-billed gull', 'peregrine falcon']
```

En el chunk siguiente, definimos como queremos que sea nuestra ontología.

```
# preparamos la ontología
# pip install Cython==0.29.28 owlready2==0.37
# pip install slugify
from owlready2 import *
from slugify import slugify
import uuid

# Creamos una ontología
onto = get_ontology("http://avistamiento-aves-3.org/onto.owl")
onto.destroy()
```

```
onto = get_ontology("http://avistamiento-aves-3.org/onto.owl")

with onto:
  class label(DataProperty):
    range = [str]
  class text_field(DataProperty):
    range = [str]
  class match_text(DataProperty):
    range = [str]
  class match_score(DataProperty):
    range = [float]
  class dbpedia(DataProperty):
    range = [str]
  class date(DataProperty):
    range = [str]
  class position(DataProperty):
    range = [str]
  class generic_context(DataProperty):
    range = [str]

  class Bird(Thing):
    pass
  class Sighting(Thing):
    pass

  class has_been_seen(ObjectProperty):
    domain = [Bird]
    range = [Sighting]
  class birds_found(ObjectProperty):
    domain = [Sighting]
    range = [Bird]
    inverse_property = has_been_seen

# Owl functions
def create_sighting(doc):
  sighting = Sighting(f"sighting-{uuid.uuid4()}")
  sighting.text_field.append(doc.text)
  return sighting

def register_bird(sighting, bird_key, chunk_nouns, score):
  slugify_name = slugify(bird_key, separator="_")
  bird_onto = types.new_class(slugify_name, (Bird,))
  bird_dict = birds[bird_key]
  bird_individual = bird_onto(f"{slugify_name}-{uuid.uuid4()}")
  bird_individual.label.append(bird_dict["name"])
  bird_individual.match_text.append(chunk_nouns)
  bird_individual.match_score.append(score)
  bird_individual.dbpedia.append(bird_dict["url"])
  sighting.birds_found.append(bird_individual)
  return bird_individual

def add_context(individual, name, type):
  if type == "DATE":
    individual.date.append(name)
```

```
  if type == "NORP" or type == "GPE":
    individual.position.append(name)
  else:
    individual.generic_context.append(name)
```

```
# calcular para todas las review
trace = []
for text in tqdm.tqdm(open(shorebirder_posts_file, "r").readlines()):
  doc = nlp(text)
  sighting = create_sighting(doc)

  for chunk in doc.noun_chunks:
    chunk_nouns = get_nouns(chunk)
    if chunk_nouns != None:
      (found, bird_key, score) = search_bird_dict(chunk_nouns)
      if found:
        bird_individual = register_bird(sighting, bird_key, chunk_nouns, score)
        trace.append(f"Found '{bird_key}' in '{chunk_nouns}' with score of
{score}")
        for ent in chunk.ents:
          add_context(bird_individual, str(ent), ent.label_)
      else:
        for ent in chunk.ents:
          add_context(sighting, str(ent), ent.label_)

onto.save(file="../owl/avistamiento-aves-3.xml", format = "rdfxml")
log_result = write(data_results_path, "shorebirder_results_log_3", trace)
print(f"Hemos encontrado {len(trace)} pájaros de {len(list(Bird.subclasses()))}
diferentes tipos")
print(log_result)
open(log_result, "r", encoding="utf-8").readlines()[0:2]
```

```
100%|██████████| 83/83 [00:46<00:00,  1.77it/s]

Hemos encontrado 87 pájaros de 71 diferentes tipos
../data/results/shorebirder_results_log_3.txt
```

```
["Found 'ring-billed gull' in 'ring-billed gull' with score of 100\n",
 "Found 'peregrine falcon' in 'female peregrine falcon' with score of 82\n"]
```

Primera versión funcional sin machacar a la dbpedia. Pero sigue siendo muy lento. Vamos a seguir probando.

4. Nueva pipeline entity_ruler

En esta aproximación vamos a añadir una pipe más al nlp `en_core_web_lg` pre-entrenado de spacy. Para ello necesitamos hacer una lista de todos los patterns que queramos poner. Es decir, debemos introducir los nombres de los pájaros que queremos que se detecten como patterns y añadir la nueva pipe al nlp.

```
# usando entity_ruler https://spacy.io/usage/rule-based-matching#entityruler
from spacy.lang.en import English

tag = "BIRD"

# init blank nlp
nlp = English()
nlp.tokenizer = custom_tokenizer(nlp)

# Añadir los nombres de pájaros
patterns = []
for key in bird_keys:
  bird = birds[key]
  doc = nlp(bird["name"])
  pattern = []
  for token in doc:
    pattern.append({
      "LOWER": token.lower_
    })

  patterns.append({
    "label": tag,
    "pattern": pattern,
    "id": key
  })

# add entity_ruler
ruler = nlp.add_pipe("entity_ruler")
with nlp.select_pipes(enable="tagger"):
  ruler.add_patterns(patterns)

nlp.to_disk("../models/entity_ruler/")
```

```
doc = nlp("that feeding gull flock continued to produce by sucking in passers by.
at one point a Bonaparte's gull got in on the action, and a flock of 21 common
terns appeared from the east and eventually settled into that flock.")
displacy.render(doc, jupyter=True, style="ent")
print(nlp.pipe_names)
```

that feeding gull BIRD flock continued to produce by sucking in passers by. at one point a Bonaparte's gull BIRD got in on the action, and a flock of 21 common terns appeared from the east and eventually settled into that flock.

```
['entity_ruler']
```

Clonamos el schema de la ontología del apartado anterior. A esta, tenemos que modificar la función `register_bird` porque en este caso no tenemos `name_chunks` ni un `score` de similitud.

```
# preparamos la ontología
from owlready2 import *
from slugify import slugify
import uuid

# Creamos una ontología
onto = get_ontology("http://avistamiento-aves-4.org/onto.owl")
onto.destroy()
onto = get_ontology("http://avistamiento-aves-4.org/onto.owl")

with onto:
  class label(DataProperty):
    range = [str]
  class text_field(DataProperty):
    range = [str]
  class match_text(DataProperty):
    range = [str]
  class match_score(DataProperty):
    range = [float]
  class dbpedia(DataProperty):
    range = [str]
  class date(DataProperty):
    range = [str]
  class position(DataProperty):
    range = [str]
  class generic_context(DataProperty):
    range = [str]

  class Bird(Thing):
    pass
  class Sighting(Thing):
    pass

  class has_been_seen(ObjectProperty):
    domain = [Bird]
    range = [Sighting]
  class birds_found(ObjectProperty):
    domain = [Sighting]
    range = [Bird]
```

```
    inverse_property = has_been_seen

# Owl functions
def create_sighting(doc):
  sighting = Sighting(f"sighting-{uuid.uuid4()}")
  sighting.text_field.append(doc.text)
  return sighting

def register_bird(sighting, bird_key):
  slugify_name = slugify(bird_key, separator="_")
  bird_onto = types.new_class(slugify_name, (Bird,))
  bird_dict = birds[bird_key]
  bird_individual = bird_onto(f"{slugify_name}-{uuid.uuid4()}")
  bird_individual.label.append(bird_dict["name"])
  bird_individual.dbpedia.append(bird_dict["url"])
  sighting.birds_found.append(bird_individual)
  return bird_individual

def add_context(individual, name, type):
  if type == "DATE":
    individual.date.append(name)
  if type == "NORP" or type == "GPE":
    individual.position.append(name)
  else:
    individual.generic_context.append(name)
```

```
# calcular para todas las review
nlp = spacy.load("../models/entity_ruler")
nlp.tokenizer = custom_tokenizer(nlp)

trace = []
for text in tqdm.tqdm(open(shorebirder_posts_file, "r").readlines()):
  doc = nlp(text)
  sighting = create_sighting(doc)
  for ent in doc.ents:
    if ent.label_ == tag and ent.ent_id_ != "":
      bird_individual = register_bird(sighting, ent.ent_id_)
      trace.append(f"Found '{ent.ent_id_}'")
    else:
      add_context(sighting, str(ent), ent.label_)

onto.save(file="../owl/avistamiento-aves-4.xml", format = "rdfxml")
log_result = write(data_posts_path, "shorebirder_results_log_4", trace)
print(f"Hemos encontrado {len(trace)} pájaros de {len(list(Bird.subclasses()))}
diferentes tipos")
print(log_result)
open(log_result, "r", encoding="utf-8").readlines()[0:5]
```

```
100%|██████████| 83/83 [00:01<00:00, 59.50it/s]


Hemos encontrado 52 pájaros de 45 diferentes tipos
../data/posts/shorebirder_results_log_4.txt
```

```
["Found 'gull'\n",
 "Found 'ring-billed gull'\n",
 "Found 'peregrine falcon'\n",
 "Found 'gull'\n",
 "Found 'gull'\n"]
```

Mejoramos mucho la velocidad por una cantidad aceptable de reconocimiento de aves. Pero no nos detecta todas las que conseguimos detectar en el `intento 3`. Esto es porque la capa de entity_ruler no es parte del modelo sino pattern matching. por ello no es capaz de encontrar las formas en plural de las aves que el intento 3 si encuentra. Al tampoco usar `ner` perdemos la capacidad de detectar elementos del contexto.

5. Entrenar ner Spacy

En este apartado vamos a centrarnos en mejorar la detección de spacy entrenando un modelo de clasificación. Las frases de entreno las cogeremos de las descripciones de la dbpedia. Los datos de entreno deberán estar compuestos de una frase más las posiciones de los nombres de pájaros. Para reconocer los pájaros que aparecen en las descripciones usaremos el `entity ruler` del apartado anterior. Al usar el entity ruler para esta tarea vamos a suponer que en la dbpedia no hay faltas de ortografía en las descripciones para que los pájaros sean identificables, y su nombre aparezca al menos una vez.

La pipeline de spacy que se encarga de etiquetar las entities se llama Entity Recognizer o `ner`. Según la documentación de spacy sobre como entrenar un modelo, primero necesitamos las frases de entreno correctamente etiquetadas, vectores por palabra a etiquetar y un fichero de configuración con las pipelines a entrenar.

La generación de frases de entreno usaremos el 100% de frases para train y 20% para test (subconjunto de train). He decidido "correr el riesgo" de over-fitting del modelo a la de no registrar pájaros al modelo. He tomado esta decisión dada las escasas frases/pájaro que podemos obtener de la dbpedia.

Para que ner pueda reconocer y comparar las aves vamos a tener que generar vectores. Sabemos que el nlp de spacy no genera vectores para palabras fuera del modelo, o en el caso de `en_core_web_sm` directamente no tiene. Para ello, la generación de vectores se va a realizar mediante la librería gensim y la técnica Word2Vec como sugieren en la documentación de spacy.

El fichero de configuración guardado en `src/birds_config.cfg` se ha generado usando la herramienta web que proporciona spacy seleccionando `ner` + `cpu` + `efficiency`. Ner, porque es la pipeline que queremos entrenar. Cpu, porque he tenido problemas para compilar pytorch para usar la gpu y es una opción más compatible. Efficiency porque nosotros le proporcionaremos los vectores.

```
# Transformar las descripciones en datos de entreno y test para spacy
import gensim
import random
import unicodedata
def strip_accents(s):
   return ''.join(c for c in unicodedata.normalize('NFD', s)
                  if unicodedata.category(c) != 'Mn')

# init the "base nlp"
nlp = spacy.load("en_core_web_lg")
# add custom tokenizer
nlp.tokenizer = custom_tokenizer(nlp)
# add entity_ruler
entity_ruler = spacy.load("../models/entity_ruler")
nlp.add_pipe("entity_ruler", source=entity_ruler, before="ner")

def get_vector(ent):
  if ent.has_vector:
    return ent.vector
  else:
    return gensim.models.Word2Vec(str(ent), min_count = 1, window = 6, sg=0)

# build training sentences
training_data = [
  # ("Tokyo Tower is 333m tall.", [(0, 11, "BUILDING")], tag/label, vector), #
example
]
for key in tqdm.tqdm(bird_keys):
  bird = birds[key]
  bird_name = bird["name"]

  if len(bird_name) == 0:
    continue

  bird_name = re.sub(remove_parentesis_text, "", str(bird_name))
  description = bird["description"]

  for train_sentence in re.split(sentence_endings, description):
    # usamos el reconocimiento de aves anterior para encontrar las aves
    # suponemos que la dbpedia tiene las aves bien identificadas
    doc = nlp(train_sentence)
    positions = []
    for ent in doc.ents:
      if ent.label_ == tag and ent.ent_id_ != "":
        vector = get_vector(ent)
        positions.append((ent.start_char, ent.end_char, ent.ent_id_, vector))

    if len(positions) > 0:
      training_data.append(
        (train_sentence, positions)
      )
```

```
def outer_join(lst1, lst2):
  lst2_names = [name for name, annotations in lst2]
  lst3 = [(name, annotations) for name, annotations in lst1 if not name in
lst2_names]
  return lst3

test_data = random.sample(training_data, k=round(len(training_data)*0.2))

# si quitamos las frases de test del modelo de entreno luego no vamos a poder
detectar esas aves... no es ideal pero tampoco podemos asegurar que tengamos
frases para todos los pájaros
training_data = outer_join(training_data, test_data)

for text, annotations in training_data[0:5]:
  print(text, [(a, b, c) for a, b, c, d in annotations])
  print("-"*10)
```

```
c:\tools\Anaconda3\lib\site-packages\spacy\language.py:710: UserWarning: [W113]
Sourced component 'entity_ruler' may not work as expected: source vectors are not
identical to current pipeline vectors.
  warnings.warn(Warnings.W113.format(name=source_name))
100%|██████████| 5/5 [00:00<00:00,  7.60it/s]

Actenoides is a genus of kingfishers in the subfamily Halcyoninae [(0, 10,
'actenoides')]
----------
 The genus Actenoides was introduced by the French ornithologist Charles Lucien
Bonaparte in 1850 [(11, 21, 'actenoides')]
----------
 The type species is Hombron's kingfisher (Actenoides hombroni) [(21, 41,
"hombron's kingfisher"), (43, 53, 'actenoides')]
----------
 A molecular study published in 2017 found that the genus Actenoides, as currently
defined, is paraphyletic [(58, 68, 'actenoides')]
----------
 The glittering kingfisher in the monotypic genus Caridonax is a member of the
clade containing the species in the genus Actenoides [(5, 26, 'glittering
kingfisher'), (121, 131, 'actenoides')]
----------
```

```
# build train set
from spacy.tokens import DocBin
from spacy.util import filter_spans

nlp = English()
# no separar por "-"
nlp.tokenizer = custom_tokenizer(nlp)
# añadir entity_ruler del apartado anterior
```

```
entity_ruler = spacy.load("../models/entity_ruler")
nlp.add_pipe("entity_ruler", source=entity_ruler)

def build_db(nlp, data):
  skips = 0
  # the DocBin will store the example documents
  db = DocBin()
  for text, annotations in tqdm.tqdm(data):
    doc = nlp(text)
    ents = []
    for start, end, label, vector in annotations:
      span = doc.char_span(start, end, label=label, vector=vector)
      if span is None:
        skips += 1
      else:
        ents.append(span)
    filtered_ents = filter_spans(ents)
    doc.ents = filtered_ents
    db.add(doc)
  return (db, skips)

(db, skips) = build_db(nlp, training_data)
print(f"Skipped {skips} entries")
db.to_disk("./birds_train.spacy")
```

```
100%|██████████| 16/16 [00:00<00:00, 187.09it/s]

Skipped 0 entries
```

```
# build test set
(db, skips) = build_db(nlp, test_data)
print(f"Skipped {skips} entries")
db.to_disk("./birds_test.spacy")
```

```
100%|██████████| 4/4 [00:00<00:00, 222.21it/s]

Skipped 0 entries
```

```
# Train the model https://spacy.io/usage/training
os.environ['KMP_DUPLICATE_LIB_OK']='True'

# train the model
!python -m spacy init fill-config birds_config.cfg config.cfg
!python -m spacy train config.cfg --output ../models/custom_ner --paths.train
```

```
./birds_train.spacy --paths.dev ./birds_test.spacy

## powershell: (launch from console to use all cpu cores)
# $env:KMP_DUPLICATE_LIB_OK = "True"
# python -m spacy init fill-config birds_config.cfg config.cfg
# python -m spacy train config.cfg --output ../models/custom_ner --paths.train
./birds_train.spacy --paths.dev ./birds_test.spacy
```

```
# juntamos el entreno previo con las patterns
nlp = spacy.load("../models/custom_ner/model-best")
nlp.tokenizer = custom_tokenizer(nlp)

# entity_ruler = spacy.load("../models/entity_ruler")
# nlp.add_pipe("entity_ruler", source=entity_ruler, before="ner")

# ner = spacy.load("en_core_web_sm") # lg tiene vectores propios por lo que no
podemos mezclar los vectores generados en nuestro modelo
# nlp.add_pipe("ner", name="base_ner", source=ner, after="ner")

# comprobamos las pipes
print(nlp.pipe_names)

# try
doc = nlp("The golden-fronteds fulvetta (Schoeniparus variegaticeps), also known
as the gold-fronted fulvetta, is a species of bird in the family Pellorneidae")
displacy.render(doc, jupyter=True, style="ent")

doc = nlp("The African goshawk (Accipiter tachiro) is a species of African bird of
prey in the genus Accipiter which is the type genus of the family Accipitridae.")
displacy.render(doc, jupyter=True, style="ent")

doc = nlp("The type species is Hombron's kingfisher (Actenoides hombroni)")
displacy.render(doc, jupyter=True, style="ent")
```

```
['tok2vec', 'ner']
```

The golden-fronteds fulvetta **grey honeyeater** (Schoeniparus variegaticeps), also known as the gold-fronted fulvetta, is a species of bird in the family Pellorneidae

The African goshawk **african goshawk** (Accipiter **accipiter** tachiro) is a species of African bird of prey in the genus Accipiter **accipiter** which is the type genus of the family Accipitridae.

The type species is Hombron's kingfisher **hombron's kingfisher** (Actenoides **actenoides** hombroni)

```
# calcular para todas las review
found_with_ruler = 0
found_with_ner = 0
error = 0
maybe_matches = []
for text in tqdm.tqdm(open(shorebirder_posts_file, "r").readlines()):
  doc = nlp(text)
  # if len(doc.ents) > 0:
  #   displacy.render(doc, jupyter=True, style="ent")
  for ent in doc.ents:
    if ent.label_ == "BIRD":
      (valid, alt_key, accuracy) = search_bird_dict(str(ent))
      if valid:
        found_with_ruler += 1
        maybe_matches.append((ent.ent_id_, str(ent)))
      else:
        error += 1
    else:
      (valid, alt_key, accuracy) = search_bird_dict(str(ent))
      if valid:
        found_with_ner += 1
        maybe_matches.append((ent.label_, str(ent)))
      else:
        error += 1

# Pintar y guardar resultado
result_lines = []
for bird_key, original in set(maybe_matches):
  bird = dict(birds[bird_key])
  name = bird["name"]
  url = bird["url"]
  result_lines.append(f"Hemos encontrado '{original}' con entrada en la dbpedia
[{name}]'{url}'.")


result_lines_file = write(data_results_path, "shorebirder_results_5",
result_lines)
print(f"Hemos encontrado {len(result_lines)} pájaros distintos de
{len(maybe_matches)}.",
  f"{found_with_ner} encontrados con ner y {found_with_ruler} encontrados con
entity ruler.",
  f"Ha habido {error} falsos casos")
open(result_lines_file, "r", encoding="utf-8").readlines()[0:5]
```

```
100%|██████████| 83/83 [00:17<00:00,  4.77it/s]

Hemos encontrado 24 pájaros distintos de 33. 33 encontrados con ner y 0
encontrados con entity ruler. Ha habido 7 falsos casos
```

```
["Hemos encontrado 'yellow-bellied' con entrada en la dbpedia
[Accipiter]'http://dbpedia.org/resource/Accipiter'.\n",
 "Hemos encontrado 'purple-crowned lorikeet' con entrada en la dbpedia [Common
kingfisher]'http://dbpedia.org/resource/Common_kingfisher'.\n",
 "Hemos encontrado 'gull' con entrada en la dbpedia
[Pycnonotus]'http://dbpedia.org/resource/Pycnonotus'.\n",
 "Hemos encontrado 'common terns' con entrada en la dbpedia [Common
kingfisher]'http://dbpedia.org/resource/Common_kingfisher'.\n",
 "Hemos encontrado 'common in' con entrada en la dbpedia [Common
kingfisher]'http://dbpedia.org/resource/Common_kingfisher'.\n"]
```

El modelo entrenado es peor que en el apartado 4. (se ha añadido una validación porque daba muchos casos falsos)

Se ha probado de unir la pipeline del apartado 4, `entity_ruler` (código comentado 2 bloques antes) y `ner` para mejorar los resultados y obtener contexto. Al no mejorarse los resultados del apartado 4 se ha abandonado el desarrollo dando el modelo del apartado 3 como el mejor.

Conclusión

El **intento 1** usando solamente los modelos pre-entrenados de spacy no son capaces de detectar aves. Previsible pero valía la pena intentar.

El **intento 2** por planteamiento (y pista en el enunciado) es muy lento y se "explota" a la dbpedia a llamadas a base de datos para verificaciones simples. Tampoco podemos decir que sea una solución válida.

El **intento 3** supone la primera solución válida al ejercicio, podría rebautizarse como *solución 1*. No solo descubre mayor número de pájaros sino que es "preciso". La pega que tiene su ejecución es la lentitud ya que requiere pasar pos el nlp de spacy para segmentar y encontrar entidades de segmento, y además, buscar si hay un nombre de pájaro realizando una comparación contra TODOS las posibles aves.

El **intento 4** también solución válida mejora en eficiencia al intento 3. En este caso, sacrificamos precisión frente a rendimiento. El modelo es notablemente más rápido, pero no detecta variaciones. Una solución, podría ser alimentar a la pipe con distintas variaciones de los nombres de pájaros como añadir su variante al plural.

El **intento 5**, posiblemente el más ambicioso y decepcionante a la vez, pretendía unir lo mejor de los intentos 3 y 4. Entrenar un modelo de clasificación. El modelo de clasificación consiste en entrenar una pipeline `ner` con los distintas frases donde "manualmente" le mostrásemos donde esta el nombre de cada pájaro y la etiqueta que le debe colocar. Además, para mejorar la detección se añadiría la pipeline del intento 4, y para obtener datos del contexto se añadiría una pipeline `ner` pre-entrenada. Las pruebas con 20-50 pájaros funcionaban increíblemente bien, tiempo de entreno largo, pero el modelo era bueno. Al entrenar con los 10369 pájaros el resultado fue muy distinto, el *entity_ruler* del intento 4 obtiene todos los match, y nuestro

ner no consigue detectar nada. La guinda sobre el pastel es que nuestro ner "obtiene" pájaros que no existen. Supongo que la peor parte fue darse cuenta que tras 16h de "training" el modelo siguiera siendo menos efectivo que las dos soluciones anteriores estas siendo más "simples".