Procesamiento de lenguaje natural

En este capítulo se verá como procesar lenguaje natural haciendo uso de la librería/herramienta *spaCy*.

En lugar de transformar, spaCy conserva el texto original a lo largo de todo el proceso, añadiendo nuevas capas de información al mismo.

Se irá mostrando paso a paso como se hace uso de las distintas herramientas que proporciona la librería con ejemplos sencillos, para después finalizar con todo el proceso aplicado sobre el Data Frame que se ha ido creando en los capítulos anteriores.

```
import sys, os

#Carga del archivo setup.py
%run -i ../pyenv_settings/setup.py

#Imports y configuraciones de gráficas
%run "$BASE_DIR/pyenv_settings/settings.py"

#Reset del entorno virtual al iniciar la ejecución
#%reset -f

%reload_ext autoreload
%autoreload 0
%config InlineBackend.figure_format = 'png'

# to print output of all statements and not just the last
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# otherwise text between $ signs will be interpreted as formula and
printed in italic
pd.set_option('display.html.use_mathjax', False)
```

```
You are working on a local system.
Files will be searched relative to "..".
```

PNL utilizando *spaCy*

Antes de utilizar spaCy

A la hora de instalar el modelo el usuario se debe asegurar de que algunas herramientas y librerías estén actualizadas a su última versión y que sean compatibles con la versión de python que se tiene instalada. En mi caso cuento con Python 3.12.7, y he tenido que actualizar con el comando *"pip install --upgrade nombre-paquete"*: *pip*, *setuptools*, *wheel*, *spacy*, *pydantic*, *thic*, y asegurarme de que las versiones eran compatibles entre ellas, por ejemplo, para thic tuve que instalar la versión 8.3.0 para asegurar la compatibilidad.

Una vez hecho todo esto, ya se puede instalar el modelo desde terminal (en este caso la del entorno virtual de python) con el comando: *python -m spacy download en_core_web_sm*

Si aún con todo esto se siguen generando errores, como es mi caso, se recomienda crear un nuevo entorno virtual con una versión de más estable, como la 3.10 o 3.11, para asegurar la compatibilidad.

Para poder contar con distintas versiones de Python, es necesario tener instalado el paquete *pyenv* en el sistema. Una vez instalado, es posible instalar la versión deseada, 3.10.9 en mi caso. A partir de este modelo, se creará un entorno virtual utilizando pyenv e indicando la versión que se desea utilizar a la hora de definirlo.

Instanciación de un pipeline

Se va a utilizar un modelo de procesamiento preentrenado, para luego instanciar un pipeline que nos servirá a lo largo del capítulo.

Cabe recalcar que el modelo utilizado, *en_core_web_sm*, debe ser descargado manualmente antes de utilizarse, en caso contrario no se cargará y el programa lanzará un error.

En la variable "nlp" se almacenará el objeto *Language*, el cuál contiene el vocabulario, el modelo y el pipeline de procesamiento.

```
import spacy

#Carga del modelo
nlp = spacy.load('en_core_web_sm')

#Instanciación del pipeline
nlp.pipeline
```

```
[('tok2vec', <spacy.pipeline.tok2vec.Tok2Vec at 0x7bedac4adc00>),
 ('tagger', <spacy.pipeline.tagger.Tagger at 0x7bedaa42b760>),
 ('parser', <spacy.pipeline.dep_parser.DependencyParser at
0x7bedaa5598c0>),
 ('attribute_ruler',
  <spacy.pipeline.attributeruler.AttributeRuler at 0x7bedaa267400>),
 ('lemmatizer',
  <spacy.lang.en.lemmatizer.EnglishLemmatizer at 0x7bedaa1c6fc0>),
 ('ner', <spacy.pipeline.ner.EntityRecognizer at 0x7bedaa559a10>)]
```

El tokenizador de spaCy es bastante rápido, pero el resto de tareas no. Es por ello que si se desea analizar un dataset considerablemente extenso, se recomienda desactivar algunas funciones del modelo para que la duración disminuya significativamente.

En este caso, se va a desactivar *parser* y *named-entity recognition* porque se va a hacer uso, por ahora, del tokenizador y el *part-of-speech tagger*, etiquetador de partes del discurso, el cuál se explicará más adelante.

```
nlp = spacy.load("en_core_web_sm", disable=["parser", "ner"])
nlp.pipeline
```

```
[('tok2vec', <spacy.pipeline.tok2vec.Tok2Vec at 0x7beda8dc9cc0>),
 ('tagger', <spacy.pipeline.tagger.Tagger at 0x7beda8dca260>),
 ('attribute_ruler',
  <spacy.pipeline.attributeruler.AttributeRuler at 0x7beda8b1cc80>),
 ('lemmatizer',
  <spacy.lang.en.lemmatizer.EnglishLemmatizer at 0x7beda8b22380>)]
```

Procesamiento de texto

En la llamada al pipeline nlp, este devuelve un objeto de tipo *spacy.tokens.doc.Doc* que contiene el acceso a los tokens, spans (rangos de tokens), y algunas anotaciones sobre el token.

```
text = "My friend is a professional race car drive, I would like to
experience that kind of feeling"
doc = nlp(text)

for token in doc:
    print(token, end="|")
```

```
My|friend|is|a|professional|race|car|drive|,|I|would|like|to|
experience|that|kind|of|feeling|
```

Se definirá a continuación una función que genera una tabla que contenga todos los tokens y sus atributos. El resultado se podría definir como un DataFrame en el que se puede utilizar la posición de cada elemento (token) como índice.

```
def display_nlp(doc, include_punct=False):
    #Generación del DataFrame con los tokens
    rows = []
    for i, t in enumerate(doc):
        if not t.is_punct or include_punct:
            row = {'token': i,  'text': t.text, 'lemma_': t.lemma_,
                   'is_stop': t.is_stop, 'is_alpha': t.is_alpha,
                   'pos_': t.pos_, 'dep_': t.dep_,
                   'ent_type_': t.ent_type_, 'ent_iob_': t.ent_iob_}
            rows.append(row)

    df = pd.DataFrame(rows).set_index('token')
    df.index.name = None

    return df
```

Cuando visualizamos el Data Frame creado, se observa que hay una columna con el atributo "is_stop" que indica si se trata de una *stop word* o no sin necesidad de usar un diccionario como se vió en capítulos anteriores gracias al uso del modelo ya preentrenado.

```
display_nlp(doc)
```

```
            text        lemma_  is_stop  is_alpha  pos_ dep_ ent_type_
\
0             My            my     True      True  PRON
1         friend        friend    False      True  NOUN
2             is            be     True      True   AUX
3              a             a     True      True   DET
4   professional  professional    False      True   ADJ
5           race          race    False      True  NOUN
6            car           car    False      True  NOUN
7          drive         drive    False      True  NOUN
9              I             I     True      True  PRON
10         would         would     True      True   AUX
11          like          like    False      True  VERB
12            to            to     True      True  PART
13    experience    experience    False      True  VERB
14          that          that     True      True   DET
15          kind          kind    False      True  NOUN
16            of            of     True      True   ADP
17       feeling          feel    False      True  VERB


   ent_iob_
0
1
2
3
4
5
6
7
9
10
11
12
13
```

```
14
15
16
17
```

Personalización del tokenizador

Debido a que la gran mayoría del texto que se va a desear analizar será inglés, hay ciertas palabras o expresiones que deben tomarse en cuenta para que el tokenizador no las elimine o separe. Este es el caso de palabras compuestas que están unidas con guión o guión bajo, expresiones o palabras que se inician con un "#" que puede dar un mayor contexto al texto, etc.

Se va a definir una función que, haciendo uso del tokenizador de spacy, lo modifica al mismo tiempo para que incluya este tipo de formaciones.

```
from spacy.tokenizer import Tokenizer
from spacy.util import compile_prefix_regex, \
                       compile_infix_regex, compile_suffix_regex

def custom_tokenizer(nlp):

    # use default patterns except the ones matched by re.search
    prefixes = [pattern for pattern in nlp.Defaults.prefixes
                if pattern not in ['-', '_', '#']]
    suffixes = [pattern for pattern in nlp.Defaults.suffixes
                if pattern not in ['_']]
    infixes  = [pattern for pattern in nlp.Defaults.infixes
                if not re.search(pattern, 'xx-xx')]

    return Tokenizer(vocab          = nlp.vocab,
                     rules          =
nlp.Defaults.tokenizer_exceptions,
                     prefix_search  =
compile_prefix_regex(prefixes).search,
                     suffix_search  =
compile_suffix_regex(suffixes).search,
                     infix_finditer =
compile_infix_regex(infixes).finditer,
                     token_match    = nlp.Defaults.token_match)

text = "@Pete: choose low-carb #food #eat-smart. _url_ ;-) 😋"

#nlp = spacy.load('en_core_web_sm')
nlp.tokenizer = custom_tokenizer(nlp)

doc = nlp(text)
for token in doc:
    print(token, end="|")
```

```
@Pete|:|choose|low-carb|#food|#eat-smart|.|_url_|;-)|😋|□|
```

Trabajando con palabras de parada (Stop Words)

Como se vio en la tabla creada antes, spaCy tiene su propio diccionario de Stop Words y clasifica cada token analizado en si se trata de una de esta clase de palabras o no.

```
nlp = spacy.load('en_core_web_sm')
text = "Dear Ryan, we need to sit down and talk. Regards, Pete"
doc = nlp(text)

#Clasificamos como NO stop word aquellas que no lo son (stop words del diccionario y signos de puntuación)
non_stop = [t for t in doc if not t.is_stop and not t.is_punct]
print(non_stop)
```

```
[Dear, Ryan, need, sit, talk, Regards, Pete]
```

Es posible que, dependiendo del objetivo del trabajo, se quieran incluir o excluir diversas palabras del diccionario en cuestión, por ello se mostrará a continuación un ejemplo de como hacerlo.

A partir de la versión 3.0 de spaCy ya no es posible modificar el diccionario de un modelo preentrenado, pero sí se puede crear una subclase para el lenguaje seleccionado

```
from spacy.lang.en import English

#Palabras que se desean incluir/excluir
excluded_stop_words = {'down'}
included_stop_words = {'dear', 'regards'}

class CustomEnglishDefaults(English.Defaults):
    #Crea una copia de la lista original de stop words
    stop_words = English.Defaults.stop_words.copy()
    #Incluye y excluye aquellas que deseamos en la copia que utilizaremos
    stop_words -= excluded_stop_words
    stop_words |= included_stop_words

class CustomEnglish(English):
    Defaults = CustomEnglishDefaults

#utilizamos en el pipeline la nueva lista de palabras creada
nlp = CustomEnglish()

text = "Dear Ryan, we need to sit down and talk. Regards, Pete"
doc = nlp.make_doc(text) # only tokenize

tokens_wo_stop = [token for token in doc ]
for token in doc:
    if not token.is_stop and not token.is_punct:
        print(token, end='|')
```

```
Ryan|need|sit|down|talk|Pete|
```

Ahora podemos modificar la lista de stop words de forma sencilla en caso de ser necesario.

Extracción de lemas basados en partes del discurso

El lema de una palabra se refiere a la raíz de la propia palabra. Lematizar un texto puede aumentar la calidad de los modelos al mismo tiempo que ahorrar tiempo y tamaño en el proceso de entrenamiento debido a que el tamaño del vocabulario será menor.

En la tabla anterior también se veía la propiedad *"lemma_"* de cada token, en este ejemplo se usará una frase cualquiera para extraer los lemas y mostrarlos por pantalla.

```
#Se vuelve a cargar el modelo original
nlp = spacy.load('en_core_web_sm')

text = "My best friend Ryan Peters likes fancy adventure games."
doc = nlp(text)

print(*[t.lemma_ for t in doc], sep='|')
```

```
my|good|friend|Ryan|Peters|like|fancy|adventure|game|.
```

Como se observa, palabras como "best", "likes" o "games" lo que se muestra es la raíz de la palabra, y no una derivación de la misma, como en este caso pueden ser superlativos o palabras en plural.

También se hará uso de otro atributo de los tokens, las etiquetas de parte del discurso. Estas etiquetas se pueden definir como abreviaturas del tipo de palabra que es cada token, así "good" se mostrará con la etiqueta "ADJ", de adjective (adjetivo).

Se utilizará el atributo *pos_* el cuál contiene la etiqueta simplificada del token para distintos usos, como en el siguiente, en el que se desea almacenar en una variable "nouns" todos los tokens que tenga como tag "NOUN" (sustantivos) y "PROPN" (sustantivos propios).

```
#En la variable nouns guardamos aquellos tokens que son sustantivos
nouns = [t for t in doc if t.pos_ in ['NOUN', 'PROPN']]
print(nouns)
```

```
[friend, Ryan, Peters, adventure, games]
```

Se puede usar una función de la librería *textacy* que sirve para extraer palabras del texto, sustantivos y adjetivos en este caso, pero funcional con cualqueir tipo de etiqueta. Con esto se consigue, adicionalmente, la posibilidad de filtrar textos con las distintas etiquetas de los tokens

```
import textacy

tokens = textacy.extract.words(doc,
            filter_stops = True,           # default -> no extrae stop
words
```

```
            filter_punct = True,           # default True -> no extrae
signos de puntuación
            filter_nums = True,            # default False -> no
extrae números
            include_pos = ['ADJ', 'NOUN'], # default None -> extraería
todas las etiquetas (pos_)
            exclude_pos = None,            # default None -> no
excluye ninguna
            min_freq = 1)                  # frecuencia mínima de
palabras

print(*[t for t in tokens], sep='|')
```

```
best|friend|fancy|adventure|games
```

También se puede definir una función que tiene la misma funcionalidad, incluyendo como parámetro los parámetros clave (**kwargs) que se deben especificar para que el programa sepa qué buscar. Queda a decisión del usuario utilizar una u otra según sus preferencias o necesidades.

```
#En esta función se extraen los lemas, no los tokens originales con la
etiqueta indicada
#Extrapolable a la extracción de cualquier tipo de token
def extract_lemmas(doc, **kwargs):
    return [t.lemma_ for t in textacy.extract.words(doc, **kwargs)]

lemmas = extract_lemmas(doc, include_pos=['ADJ', 'NOUN'])
print(*lemmas, sep='|')
```

```
good|friend|fancy|adventure|game
```

Extracción de n-grams

En el principio del proyecto se explicó qué eran los n-grams y se explicó que estos no son de mucha utilidad a la hora del análisis, pero eso era porque no se había explicado aún como extraerlos únicamente si tenían un significado real.

SpaCy ofrece una potente herramienta basada en reglas, es decir, que un n-gram la mayor parte de las veces tiene significado si las etiquetas (tipos de palabras) que las componen cumplen un cierto criterio y orden, esto junto a la extracción de frases basada en patrones de textacy puede facilitar enormemente esta tarea.

```
#Patrón del n-gram -> adjetivo seguido de uno o más sustantivos
patterns = ["POS:ADJ POS:NOUN:+"]

#Se tiene en cuenta que puede haber diferentes versiones de textacy
if textacy.__version__ < '0.11':
    spans = textacy.extract.matches(doc, patterns=patterns)
else:
```

```
    spans = textacy.extract.matches.token_matches(doc,
patterns=patterns)

print(*[s.lemma_ for s in spans], sep='|')
```

```
good friend|fancy adventure|fancy adventure game
```

Igual que en el caso anterior, se puede definir una función que realice el mismo trabajo, pero que contará con una mayor flexibilidad al estar hecha a gusto del usuario para cumplir una función específica.

```
#Función que extrae los n-grams indicados
def extract_noun_phrases(doc, preceding_pos=['NOUN'], sep='_'):
    patterns = []
    for pos in preceding_pos:
        patterns.append(f"POS:{pos} POS:NOUN:+")

    if textacy.__version__ < '0.11':
        # as in book
        spans = textacy.extract.matches(doc, patterns=patterns)
    else:
        # new textacy version
        spans = textacy.extract.matches.token_matches(doc,
patterns=patterns)

    return [sep.join([t.lemma_ for t in s]) for s in spans]

print(*extract_noun_phrases(doc, ['ADJ', 'NOUN']), sep='|')
```

```
good_friend|fancy_adventure|fancy_adventure_game|adventure_game
```

Extracción de nombres de entidades

Las entidades son nombres, generalmente propios, que se refieren a personas, lugares, organizaciones, países, etc. que pueden estar formados por uno o más tipos de palabras.

Estas entidades están representadas por objetos *Span* que también cuentan con diversas propiedades.

Como se verá a continuación, se pueden extraer estas entidades utilizando directamente *spaCy* o definiendo una función, que también usa la librería, pero es más personalizable, al mismo tiempo que se pueden mostrar por pantalla con distintos formatos.

Impresión por pantalla de las entidades que contiene *doc*

```
text = "James O'Neill, chairman of World Cargo Inc, lives in San
Francisco."
doc = nlp(text)
```

```
for ent in doc.ents:
    print(f"({ent.text}, {ent.label_})", end=" ")
```

```
(James O'Neill, PERSON) (World Cargo Inc, ORG) (San Francisco, GPE)
```

Impresión de entidades utilizando *displacy*

```
from spacy import displacy

displacy.render(doc, style='ent', jupyter=True)
```

```
<IPython.core.display.HTML object>
```

Función que extrae las entidades de un documento:

```
def extract_entities(doc, include_types=None, sep='_'):

    ents = textacy.extract.entities(doc,
             include_types=include_types,
             exclude_types=None,
             drop_determiners=True,
             min_freq=1)

    return [sep.join([t.lemma_ for t in e])+'/'+e.label_ for e in
ents]

print(extract_entities(doc, ['PERSON', 'GPE'])) #GPE: Geopolitical
Entity
```

```
["James_O'Neill/PERSON", 'San_Francisco/GPE']
```

Extracción de características en una única función

Ahora se definirá una función que unirá todo lo visto hasta ahora en una única función, la cuál extraerá todos los lemas, adjetivos, entidades, n-grams especificados, etc. del *doc* que se le pase como argumento

```
def extract_nlp(doc):
    return {
    'lemmas'          : extract_lemmas(doc, exclude_pos = ['PART',
'PUNCT', 'DET', 'PRON', 'SYM', 'SPACE'], filter_stops = False),
    'adjs_verbs'      : extract_lemmas(doc, include_pos = ['ADJ',
'VERB']),
    'nouns'           : extract_lemmas(doc, include_pos = ['NOUN',
'PROPN']),
    'noun_phrases'    : extract_noun_phrases(doc, ['NOUN']),
    'adj_noun_phrases': extract_noun_phrases(doc, ['ADJ']),
    'entities'        : extract_entities(doc, ['PERSON', 'ORG', 'GPE',
```

```
'LOC'])
    }
```

Utilización de spaCy en un Dataset

Ahora se mostrará como realizar todas las acciones (tokenizar, extracción de características, etc) vistas hasta el momento sobre un Dataset, en este caso el creado en el anterior capítulo con todos los comentarios de un repositorio de Github.

```
#Conexión con la base de datos en la que tenemos guardado el Data
Frame
db_name = "../data/zigbee2mqtt_comments.db"
con = sqlite3.connect(db_name)
df = pd.read_sql("select * from comments", con)
con.close()

#En cada entrada de la columna texto incluimos el usuario y el
comentario que le corresponde
#df['text'] = df['user'] + ': ' + df['text']
```

Añadiremos en el Data Frame las nuevas columnas que corresponderán con las características extraídas del texto antes de ejecutar ninguna función.

```
nlp_columns = list(extract_nlp(nlp.make_doc('')).keys())
print(nlp_columns)
```

```
['lemmas', 'adjs_verbs', 'nouns', 'noun_phrases', 'adj_noun_phrases',
'entities']
```

```
for col in nlp_columns:
    df[col] = None

#Indicamos que el programa use la GPU en caso de que se disponga de
ella
#La ejecución será más rápida que en una CPU
if spacy.prefer_gpu():
    print("Working on GPU.")
else:
    print("No GPU found, working on CPU.")
```

```
No GPU found, working on CPU.
```

Se hará uso del mismo modelo preentrenado, junto con el tokenizador personalizado que se definió al principio del capítulo.

```
nlp = spacy.load('en_core_web_sm', disable=[])
nlp.tokenizer = custom_tokenizer(nlp) #opcional, puede usarse el
proporcionado por la librería directamente
```

Para procesar grandes datasets es recomendable el uso de procesamiento por lotes para un mayor rendimiento y disminuir el tiempo de ejecución.

SpaCy toma el tamaño del lote definido por el usuario, toma el mismo número de textos y los procesa internamente, añadiendo al Doc de forma iterativa los distintos lotes en el mismo orden que los datos de entrada.

```
#Antes de extraer las características, crearemos las columnas que
almacenará las de cada texto
#Crear las columnas basadas en las claves que devuelve extract_nlp
sample_doc = next(nlp.pipe(df['text'].iloc[:1]))
new_columns = extract_nlp(sample_doc).keys()
for col in new_columns:
    df[col] = None  # Inicializa las columnas en el DataFrame con
valores vacíos

batch_size = 50
batches = math.ceil(len(df) / batch_size) ###

for i in tqdm(range(0, len(df), batch_size), total=batches):
    docs = nlp.pipe(df['text'][i:i+batch_size])

    for j, doc in enumerate(docs):
        for col, values in extract_nlp(doc).items():
            df[col].iloc[i+j] = values

100%|██████████| 54/54 [00:50<00:00,  1.07it/s]

df[['text', 'lemmas', 'nouns', 'noun_phrases', 'entities']].sample(5)


text  \
538
Looks like kirovilya's fix is working then. Good to know, thank you
for the feedback!
1476  > > Zigbee2MQTT:error 2022-07-31 12:08:28: Error: network
commissioning timed out - most likely network with the same panId or
extendedPanId already exists nearby\r\n> \r\n> It looks that
another ...
860   I can confirm - after updating to newer version it fails to
autodiscovery devices.\r\n\r\n```\r\n`Exception in async_discover when
dispatching 'mqtt_discovery_new_select_mqtt': ({'availability': [...

304   For the people here whose problem was fixed by removing
"availability: true"\r\n\r\nSeems like there is a fix on the dev
branch. \r\nhttps://github.com/Koenkk/zigbee2mqtt/pull/23316\r\n\r\
nOther t...
1529
This issue is stale because it has been open 30 days with no activity.
Remove stale label or comment or this will be closed in 7 days
```

```
lemmas  \
538
[look, like, kirovilya, fix, be, work, then, good, know, thank, for,
feedback]
1476  [>, >, Zigbee2MQTT, error, 2022, 07, 31, 12:08:28, error,
network, commission, time, out, most, likely, network, with, same,
panid, or, extendedpanid, already, exist, nearby, >, look, that,
networ...
860   [can, confirm, after, update, to, new, version, fail,
autodiscovery, device, exception, in, when, dispatch,
mqtt_discovery_new_select_mqtt, availability, topic, zigbee2mqtt,
bridge, state, command...
304   [for, people, here, problem, be, fix, by, remove, availability,
true, seem, like, be, fix, on, dev, branch, https://github.com,
Koenkk, zigbee2mqtt, pull/23316, other, thread, with, believe, be,
s...
1529
[issue, be, stale, because, have, be, open, 30, day, with, activity,
remove, stale, label, or, comment, or, will, be, close, in, 7, day]


nouns  \
538
[kirovilya, fix, feedback]
1476
[Zigbee2MQTT, error, error, network, network, panid, extendedpanid,
network, channel, panid, zigbee, device, zigbee, device,
clarification]
860   [version, autodiscovery, device, mqtt_discovery_new_select_mqtt,
availability, topic, bridge, state, Garden_Sensor2, set,
temperature_unit, device, configuration_url, hassio_ingress,
kIMye0iSR_ZQq...
304                                          [people, problem,
availability, fix, dev, branch, Koenkk, pull/23316, thread, problem,
Koenkk, issues/23266, thread, #23316, issue, group, configuration,
problem]
1529
[issue, day, activity, label, comment, day]


noun_phrases  \
538
[]
1476
[]
860   [autodiscovery_device, soil_sensor, raise_e, path_=,
path_=_path, =_path, v_=, __call___return, __call___return,
url_trough, url_trough_frontend, trough_frontend, env_assignment]
```

```
304
[dev_branch, configuration_problem]
1529
[]
```

```
                                                                entities
538                                                                   []
1476                                                                  []
860   [TuYa/PERSON, MultipleInvalid/ORG, congfiguration.yaml/ORG]
304                                                                   []
1529                                                                  []
```

Una vez completado todo el proceso, guardamos el resultado de nuevo en la base de datos.

```
df[nlp_columns] = df[nlp_columns].applymap(lambda items: '
'.join(items))

con = sqlite3.connect(db_name)
df.to_sql("posts_nlp", con, index=False, if_exists="replace")
con.close()
```

```
2678
```

```
print(df.columns)
```

```
Index(['id', 'user', 'text', 'impurity', 'clean_text',
'normalized_text',
       'tokens', 'lemmas', 'adjs_verbs', 'nouns', 'noun_phrases',
       'adj_noun_phrases', 'entities'],
      dtype='object')
```