Procesamiento de lenguaje natural

NLTK (*Natural Language Tool Kit*) es un módulo para el procesamiento de lenguaje natural. Está muy desarrollado para la lengua inglesa, y menos en español.

Abordamos una introducción de los conceptos básicos, intentando ceñirnos al español en la medida de lo posible.

In [1]:

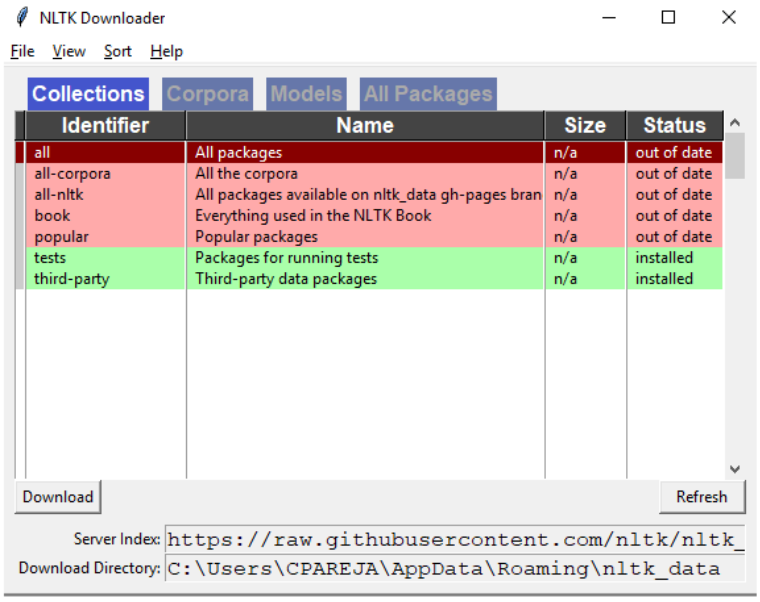
```
# Descargamos la librería y uno de los libros que trae, text1, que es Moby Dick:

import nltk
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/inde
x.xml (https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml)
```

Out[1]:

```
True
```

Al importar la librería se abre una ventana con una aplicacíon para descargar los paquetes que necesitemos:



Descarga de textos

Hay muchas fuentes de las que se pueden descargar textos para alnalizar. una de ellas es la propia librería *nltk*:

In [2]:

```
# Descargamos un libro de ese módulo:

from nltk.book import text1 # Moby Dick

print(text1)
print(text1[0:100])
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
<Text: Moby Dick by Herman Melville 1851>
['[', 'Moby', 'Dick', 'by', 'Herman', 'Melville', '1851', ']', 'ETYMOLOGY',
'.', '(', 'Supplied', 'by', 'a', 'Late', 'Consumptive', 'Usher', 'to', 'a',
'Grammar', 'School', ')', 'The', 'pale', 'Usher', '--', 'threadbare', 'in',
'coat', ',', 'heart', ',', 'body', ',', 'and', 'brain', ';', 'I', 'see', 'hi
m', 'now', '.', 'He', 'was', 'ever', 'dusting', 'his', 'old', 'lexicons', 'a
nd', 'grammars', ',', 'with', 'a', 'queer', 'handkerchief', ',', 'mockingl
y', 'embellished', 'with', 'all', 'the', 'gay', 'flags', 'of', 'all', 'the',
'known', 'nations', 'of', 'the', 'world', '.', 'He', 'loved', 'to', 'dust',
'his', 'old', 'grammars', ';', 'it', 'somehow', 'mildly', 'reminded', 'him',
'of', 'his', 'mortality', '.', '"', 'While', 'you', 'take', 'in', 'hand', 't
o', 'school', 'others', ',']
```

Una segunda fuente repleta de textos es Internet: cualquier página, en una url cualquiera, puede ser descagada para analizar su contenido:

In [3]:

```
import urllib.request

archivo = urllib.request.urlopen("http://antares.sip.ucm.es/cpareja/")
con_etiquetas = archivo.read()

from bs4 import BeautifulSoup
texto_limpio = BeautifulSoup(con_etiquetas, "lxml")
texto = texto_limpio.get_text(strip=True)

# Veamos un fragmento del texto limpio de etiquetas:

texto[504:887]
```

Out[3]:

```
'Áreas e intereses principales:\r\n                    Programación funciona
l, métodos formales,\r\n                    transformación de programas.\r\n
Entornos de programación y herramientas de visualización.\r\n
Testing and performance evaluation, y Cloud computing.\r\n\t\t\t\t\tEnseñanz
a de la Informática.\r\n                    Historia de las matemáticas.'
```

No podemos dejar de citar un tercer repositorio de libros: la librería Gutenberg.

In [4]:

```
from urllib import request

# Descarga de un texto:

url_1001_noches_part_1 = "http://www.gutenberg.org/cache/epub/47287/pg47287.txt"
respuesta = request.urlopen(url_1001_noches_part_1)
mil_y_una_noches = respuesta.read().decode('utf8')

print(type(mil_y_una_noches))
print(len(mil_y_una_noches))
print("-----------------------------------------------------")
fragmento_1001_noches = mil_y_una_noches[40150:40582]
print(fragmento_1001_noches)
```

```
<class 'str'>
396401
-----------------------------------------------------
Pero después me acordé de la joya que te
destinaba y que te di al llegar á tu palacio. Volví, pues, y encontré á
mi mujer acostada con un esclavo negro, durmiendo en los tapices de mi
cama. Los maté á los dos, y vine hacia ti, muy atormentado por el
recuerdo de tal aventura. Este fué el motivo de mi primera palidez y de
mi enflaquecimiento. En cuanto á la causa de haber recobrado mi buen
color, dispénsame de mencionarla.»
```

Tokens

En ocasiones, el texto descargado viene ya troceado en una lista de elementos léxicos, como es el caso de Moby Dick: palabras (como 'Moby') y símbolos (como "."). Cada uno de ellos es un *token*.

En otras ocasiones, tenemos que hacerlo nosotros. La separación de los tokens es normalmente el primer paso para examinar un texto, ya sea de lenguaje natural o de un lenguaje formal, como son los lenguajes de programación o el lenguaje algebraico.

Vemos que el text1 viene ya separado en tokens, pero si deseamos trabajar con un texto traído de cualquier otra fuente, es posible que no venga separado en tokens. Hay que *tokenizarlo*, si queremos usar este anglicismo tan extendido:

In [5]:

```
from nltk.tokenize import word_tokenize

print(fragmento_1001_noches)

print("-----------------------------------------------------")

fragmento_1001_noches_tokens = word_tokenize(fragmento_1001_noches)

print(fragmento_1001_noches_tokens)
```

```
Pero después me acordé de la joya que te
destinaba y que te di al llegar á tu palacio. Volví, pues, y encontré á
mi mujer acostada con un esclavo negro, durmiendo en los tapices de mi
cama. Los maté á los dos, y vine hacia ti, muy atormentado por el
recuerdo de tal aventura. Este fué el motivo de mi primera palidez y de
mi enflaquecimiento. En cuanto á la causa de haber recobrado mi buen
color, dispénsame de mencionarla.»
-----------------------------------------------------
['Pero', 'después', 'me', 'acordé', 'de', 'la', 'joya', 'que', 'te', 'destin
aba', 'y', 'que', 'te', 'di', 'al', 'llegar', 'á', 'tu', 'palacio', '.', 'Vo
lví', ',', 'pues', ',', 'y', 'encontré', 'á', 'mi', 'mujer', 'acostada', 'co
n', 'un', 'esclavo', 'negro', ',', 'durmiendo', 'en', 'los', 'tapices', 'd
e', 'mi', 'cama', '.', 'Los', 'maté', 'á', 'los', 'dos', ',', 'y', 'vine',
'hacia', 'ti', ',', 'muy', 'atormentado', 'por', 'el', 'recuerdo', 'de', 'ta
l', 'aventura', '.', 'Este', 'fué', 'el', 'motivo', 'de', 'mi', 'primera',
'palidez', 'y', 'de', 'mi', 'enflaquecimiento', '.', 'En', 'cuanto', 'á', 'l
a', 'causa', 'de', 'haber', 'recobrado', 'mi', 'buen', 'color', ',', 'dispén
same', 'de', 'mencionarla', '.', '»']
```

Otra forma de separación que se necesita con frecuencia es por frases. También está disponible en la librería *nltk.tokenize*:

In [6]:

```
from nltk.tokenize import sent_tokenize

print(fragmento_1001_noches)

print("-----------------------------------------------------")

fragmento_1001_noches_frases = sent_tokenize(fragmento_1001_noches)

print(fragmento_1001_noches_frases)
```

```
Pero después me acordé de la joya que te
destinaba y que te di al llegar á tu palacio. Volví, pues, y encontré á
mi mujer acostada con un esclavo negro, durmiendo en los tapices de mi
cama. Los maté á los dos, y vine hacia ti, muy atormentado por el
recuerdo de tal aventura. Este fué el motivo de mi primera palidez y de
mi enflaquecimiento. En cuanto á la causa de haber recobrado mi buen
color, dispénsame de mencionarla.»
-----------------------------------------------------
['Pero después me acordé de la joya que te\r\ndestinaba y que te di al llega
r á tu palacio.', 'Volví, pues, y encontré á\r\nmi mujer acostada con un esc
lavo negro, durmiendo en los tapices de mi\r\ncama.', 'Los maté á los dos, y
vine hacia ti, muy atormentado por el\r\nrecuerdo de tal aventura.', 'Este f
ué el motivo de mi primera palidez y de\r\nmi enflaquecimiento.', 'En cuanto
á la causa de haber recobrado mi buen\r\ncolor, dispénsame de mencionarl
a.»']
```

Stopwords, palabras vacías o palabras comunes

Las *stopwords* son palabras que no aportan significado por sí solas. Típicamente, se trata de los artículos, preposiciones, conjunciones y pronombres, aunque también algunos verbos.

Normalmente, estas palabras se descartan en muchas aplicaciones de análisis de lenguaje natural, pues las aplicaciones más frecuentes extraen información de las palabras con significado intrínseco, tales como sutantivos y adjetivos.

Veamos cuáles son las palabras vacías en español que vienen ya definidas en la librería *nltk*.

In [7]:

```
from nltk.corpus import stopwords

print(type(stopwords))

stop_espannol = stopwords.words('spanish')
print(len(stop_espannol))
print(stop_espannol)
```

```
<class 'nltk.corpus.util.LazyCorpusLoader'>
313
['de', 'la', 'que', 'el', 'en', 'y', 'a', 'los', 'del', 'se', 'las', 'por',
'un', 'para', 'con', 'no', 'una', 'su', 'al', 'lo', 'como', 'más', 'pero',
'sus', 'le', 'ya', 'o', 'este', 'sí', 'porque', 'esta', 'entre', 'cuando',
'muy', 'sin', 'sobre', 'también', 'me', 'hasta', 'hay', 'donde', 'quien', 'd
esde', 'todo', 'nos', 'durante', 'todos', 'uno', 'les', 'ni', 'contra', 'otr
os', 'ese', 'eso', 'ante', 'ellos', 'e', 'esto', 'mí', 'antes', 'algunos',
'qué', 'unos', 'yo', 'otro', 'otras', 'otra', 'él', 'tanto', 'esa', 'estos',
'mucho', 'quienes', 'nada', 'muchos', 'cual', 'poco', 'ella', 'estar', 'esta
s', 'algunas', 'algo', 'nosotros', 'mi', 'mis', 'tú', 'te', 'ti', 'tu', 'tu
s', 'ellas', 'nosotras', 'vosotros', 'vosotras', 'os', 'mío', 'mía', 'míos',
'mías', 'tuyo', 'tuya', 'tuyos', 'tuyas', 'suyo', 'suya', 'suyos', 'suyas',
'nuestro', 'nuestra', 'nuestros', 'nuestras', 'vuestro', 'vuestra', 'vuestro
s', 'vuestras', 'esos', 'esas', 'estoy', 'estás', 'está', 'estamos', 'estái
s', 'están', 'esté', 'estés', 'estemos', 'estéis', 'estén', 'estaré', 'estar
ás', 'estará', 'estaremos', 'estaréis', 'estarán', 'estaría', 'estarías', 'e
staríamos', 'estaríais', 'estarían', 'estaba', 'estabas', 'estábamos', 'esta
bais', 'estaban', 'estuve', 'estuviste', 'estuvo', 'estuvimos', 'estuvistei
s', 'estuvieron', 'estuviera', 'estuvieras', 'estuviéramos', 'estuvierais',
'estuvieran', 'estuviese', 'estuvieses', 'estuviésemos', 'estuvieseis', 'est
uviesen', 'estando', 'estado', 'estada', 'estados', 'estadas', 'estad', 'h
e', 'has', 'ha', 'hemos', 'habéis', 'han', 'haya', 'hayas', 'hayamos', 'hayá
is', 'hayan', 'habré', 'habrás', 'habrá', 'habremos', 'habréis', 'habrán',
'habría', 'habrías', 'habríamos', 'habríais', 'habrían', 'había', 'habías',
'habíamos', 'habíais', 'habían', 'hube', 'hubiste', 'hubo', 'hubimos', 'hubi
steis', 'hubieron', 'hubiera', 'hubieras', 'hubiéramos', 'hubierais', 'hubie
ran', 'hubiese', 'hubieses', 'hubiésemos', 'hubieseis', 'hubiesen', 'habiend
o', 'habido', 'habida', 'habidos', 'habidas', 'soy', 'eres', 'es', 'somos',
'sois', 'son', 'sea', 'seas', 'seamos', 'seáis', 'sean', 'seré', 'serás', 's
erá', 'seremos', 'seréis', 'serán', 'sería', 'serías', 'seríamos', 'seríai
s', 'serían', 'era', 'eras', 'éramos', 'erais', 'eran', 'fui', 'fuiste', 'fu
e', 'fuimos', 'fuisteis', 'fueron', 'fuera', 'fueras', 'fuéramos', 'fuerai
s', 'fueran', 'fuese', 'fueses', 'fuésemos', 'fueseis', 'fuesen', 'sintiend
o', 'sentido', 'sentida', 'sentidos', 'sentidas', 'siente', 'sentid', 'teng
o', 'tienes', 'tiene', 'tenemos', 'tenéis', 'tienen', 'tenga', 'tengas', 'te
ngamos', 'tengáis', 'tengan', 'tendré', 'tendrás', 'tendrá', 'tendremos', 't
endréis', 'tendrán', 'tendría', 'tendrías', 'tendríamos', 'tendríais', 'tend
rían', 'tenía', 'tenías', 'teníamos', 'teníais', 'tenían', 'tuve', 'tuvist
e', 'tuvo', 'tuvimos', 'tuvisteis', 'tuvieron', 'tuviera', 'tuvieras', 'tuvi
éramos', 'tuvierais', 'tuvieran', 'tuviese', 'tuvieses', 'tuviésemos', 'tuvi
eseis', 'tuviesen', 'teniendo', 'tenido', 'tenida', 'tenidos', 'tenidas', 't
ened']
```

Limpieza: sin stopwords

En el análisis de texos, normalmente no se necesitan las stopwords: estorban. Por eso, es frecuente una operación de limpieza como la siguiente:

In [8]:

```
from nltk.corpus import stopwords

def removeStopwords(palabras):
     return [word for word in palabras if word not in stopwords.words('spanish')]

fragmento_2001_sin_stop = removeStopwords(fragmento_1001_noches_tokens)

print(fragmento_2001_sin_stop)
```

```
['Pero', 'después', 'acordé', 'joya', 'destinaba', 'di', 'llegar', 'á', 'pal
acio', '.', 'Volví', ',', 'pues', ',', 'encontré', 'á', 'mujer', 'acostada',
'esclavo', 'negro', ',', 'durmiendo', 'tapices', 'cama', '.', 'Los', 'maté',
'á', 'dos', ',', 'vine', 'hacia', ',', 'atormentado', 'recuerdo', 'tal', 'av
entura', '.', 'Este', 'fué', 'motivo', 'primera', 'palidez', 'enflaquecimien
to', '.', 'En', 'cuanto', 'á', 'causa', 'haber', 'recobrado', 'buen', 'colo
r', ',', 'dispénsame', 'mencionarla', '.', '»']
```

Frecuencia de palabras significativas

Y ahora se puede hacer algún análisis. La librería *nltk* permite contar la frecuencia de las palabras de un texto.

Poca gracia tendría esto son un fragmento breve, así que lo hacemos con el texto completo de las 1001 noches, aunque ya te advierto que el proceso llevará un tiempo...

Y mostramos los 25 tokens más repetidos.

In [9]:

```
mil_y_una_noches_tokens = word_tokenize(mil_y_una_noches)
mil_y_una_noches_tokens_sin_stop = removeStopwords(mil_y_una_noches_tokens)
frecuencias_terminos_1001_noches = nltk.FreqDist(mil_y_una_noches_tokens_sin_stop)
terms_frecs = list(frecuencias_terminos_1001_noches.items())
terms_frecs.sort(key=lambda par: par[1], reverse=True)

for termino, frec in terms_frecs[:25]:
    print(termino + " -> " + str(frec))
```

```
, -> 5198
. -> 2201
á -> 1446
: -> 1083
Y -> 949
« -> 910
! -> 900
» -> 897
dijo -> 403
rey -> 321
Entonces -> 281
? -> 235
joven -> 198
Alah -> 196
_ -> 186
Pero -> 181
[ -> 177
] -> 177
; -> 176
¡Oh -> 164
the -> 162
-- -> 162
El -> 138
¡oh -> 127
efrit -> 125
```

Es evidente que se ha de limpiar más, eliminando tokens que no tienen letras, eliminando los caracteres no letras al inicio y al final... pero esto no es importante en este momento.

Para ver el resultado de nuestro análisis, no hay nada mejor que un gráfico. Para el mismo, también nos quedamos con los 25 términos más repetidos:

In [10]:

```
import matplotlib.pyplot as plt
import numpy as np

abcisas = [x for (x,_y) in terms_frecs[:25]]
valores = [y for (_, y) in terms_frecs[:25]]

x_pos = np.arange(len(abcisas))

plt.plot(x_pos, valores)
plt.xticks(x_pos, abcisas, rotation=60)

plt.ylabel('Frecuencias')
plt.xlabel('Términos o tokens')
plt.show()
```

```
<Figure size 640x480 with 1 Axes>
```

Sinónimos y antónimos

Dentro de la librería *nltk*, el módulo *wordnet* incluye un diccionario de sinónimos y antónimos. Al instalar nltk, uno de los paquetes que se podía incluir era *wordnet*; si en su momento no lo instalaste, ahora es el momento.

El problema es que a fecha de hoy, no existe este módulo para la lengua española :-(

In [11]:

```
from nltk.corpus import wordnet

sinonimos = wordnet.synsets("big")

print(sinonimos)

print(sinonimos[0].definition())
print(sinonimos[0].examples())
```

```
[Synset('large.a.01'), Synset('big.s.02'), Synset('bad.s.02'), Synset('big.
s.04'), Synset('big.s.05'), Synset('big.s.06'), Synset('boastful.s.01'), Syn
set('big.s.08'), Synset('adult.s.01'), Synset('big.s.10'), Synset('big.s.1
1'), Synset('big.s.12'), Synset('big.s.13'), Synset('big.r.01'), Synset('boa
stfully.r.01'), Synset('big.r.03'), Synset('big.r.04')]
above average in size or number or quantity or magnitude or extent
['a large city', 'set out for the big city', 'a large sum', 'a big (or larg
e) barn', 'a large family', 'big businesses', 'a big expenditure', 'a large
number of newspapers', 'a big group of scientists', 'large areas of the worl
d']
```

In [12]:

```
sinonimos, antonimos = [], []

def sinonimos_de(palabra):
    sinonimos = [lemma.name() for
                 sin in wordnet.synsets(palabra)
                 for lemma in sin.lemmas()]
    return sinonimos

def antonimos_de(palabra):
    antonimos = [lemma.antonyms()[0].name()
                 for sin in wordnet.synsets(palabra)
                 for lemma in sin.lemmas() if lemma.antonyms()]
    return antonimos

print(sinonimos_de("big"))
print(antonimos_de("big"))
```

```
['large', 'big', 'big', 'bad', 'big', 'big', 'big', 'large', 'prominent', 'b
ig', 'heavy', 'boastful', 'braggart', 'bragging', 'braggy', 'big', 'cock-a-h
oop', 'crowing', 'self-aggrandizing', 'self-aggrandising', 'big', 'swelled',
'vainglorious', 'adult', 'big', 'full-grown', 'fully_grown', 'grown', 'grown
up', 'big', 'big', 'large', 'magnanimous', 'big', 'bighearted', 'bounteous',
'bountiful', 'freehanded', 'handsome', 'giving', 'liberal', 'openhanded', 'b
ig', 'enceinte', 'expectant', 'gravid', 'great', 'large', 'heavy', 'with_chi
ld', 'big', 'boastfully', 'vauntingly', 'big', 'large', 'big', 'big']
['small', 'little', 'small']
```

Derivación regresiva, o *steming*

Las palabras *working*, *works* y *worked* tienen la misma raíz, y es conveniente considerarlas como en la misma categoría semántica a efectos de análisis de texto. La derivación regresiva (*stem*, en inglés) nos da la raíz de una palabra, y el algoritmo más empleado para ello, en el idioma inglés, es el de Porter:

In [13]:

```
from nltk.stem import PorterStemmer

stemmer = PorterStemmer()

print(stemmer.stem('working'))
print(stemmer.stem('worked'))
print(stemmer.stem('works'))
```

```
work
work
work
```

La derivación regresiva de palabras no inglesas hay que acudir a otros paquetes, como *SnowballStemmer*, que contiene 13 idiomas además del inglés.

In [14]:

```
from nltk.stem import SnowballStemmer
raiz_espannola = SnowballStemmer("spanish")

print(raiz_espannola.stem('trabajaba'))
print(raiz_espannola.stem('trabajos'))
print(raiz_espannola.stem('trabajoso'))
print(raiz_espannola.stem('trabajaré'))
print(raiz_espannola.stem('trabajar'))
print(raiz_espannola.stem('trabajando'))
print(raiz_espannola.stem('trabajaríamos'))
```

```
trabaj
trabaj
trabaj
trabaj
trabaj
trabaj
trabaj
```

Referencias

- https://likegeeks.com/es/tutorial-de-nlp-con-python-nltk/ (https://likegeeks.com/es/tutorial-de-nlp-con-python-nltk/)
- https://www.nltk.org/book/ch01.html (https://www.nltk.org/book/ch01.html)
- https://pmoracho.github.io/blog/2017/01/04/NLTK-mi-tutorial/ (https://pmoracho.github.io/blog/2017/01/04/NLTK-mi-tutorial/)