Federico Ortega Riba

Revisión de programación explorando WordNet

Objetivos didácticos:

1.- revisar los conceptos básicos de programación en python aprendidos en el primer cuatrimestre

2.- conocer WordNet y saber acceder al conocimiento que contiene mediante consultas sencillas programadas en python

Paso 0. Para empezar necesitamos responder a estas tres cuestiones

¿Qué es wordnet?

¿Cómo está estructurado WordNet (componentes y relaciones)? (Sugerencia: en grupo preparad un mapa en papel con las relaciones de una palabra, por ejemplo la palabra 'car', con el resto de las palabras)

¿Puedo utilizar WordNet para extraer conocimiento léxico? ¿Cómo se accede a WordNet? (Sugerencia: consultar http://www.nltk.org/howto/wordnet.html)

Puedo descargar wordnet utilizando, por ejemplo, nltk:

import nltk

nltk.download('wordnet')

from nltk.corpus import wordnet as wn from nltk.corpus import wordnet as wn

Paso 1. Obtener los significados de 'car'. Guardarlos en la variable significados_car

Responde a las cuestiones siguientes:

¿qué método se utiliza?
synset()
¿qué estructura de datos se utiliza para agrupar los significados (salida)?
Una lista
¿cómo se accede al primer significado, al segundo y al último?

> Dado que los significados están organizados en forma de lista, para acceder a cualquiera de ellos basta con indicar la posición entre corchetes. Para el primero se pondría [0], para el segundo [1] y para el último [-1].

¿de qué tipo de datos es cada significado de 'car'?
¿cuántos significados tiene 'car'? cinco
¿cómo se identifica cada synset?
Cada significado se identifica mediante un lema que tiene la siguiente estructurua : nombre_lema,categoría_lema.número_significado. Poe ejemplo: 'car.n.01'
¿de qué tipo de datos es cada synset?
Un objeto de la clase Synset
--> Revisar qué es una lista (cómo se crea, cómo se accede, métodos y funciones)

--> Revisar qué es una tupla (cómo se crea, cómo se accede, métodos y funciones, diferencia con listas)
--> Revisar qué es un conjunto (cómo se crea, cómo se accede, métodos y funciones, diferencia con listas y tuplas)

Copia el código una vez que compruebes que funciona en este cuadro

```
sig_car = wn.synsets('car')
sig_car
type(sig_car[0])
str(wn.synsets('car')[0].name())
wn.synsets('WORD')
wn.synsets('WORD', pos=wn.VERB)
#A synset is identified with a 3-part name of the form: word.pos.num
```

Paso 2. Obtener el identificador del primer significado de 'car'. Comprobar si tiene categoría nombre

--->Revisar las instrucciones condicionales en python

Se puede resolver de varias formas:

1) Usando expresiones regulares

2) Utilizando el atributo pos que devuelve True o False

3) comprobando si la 'n' está en el identificador, pero esta solución tiene un fallo ¿cuál?

```
import re
reg_ex = '\.n\.'
parser = re.compile(reg_ex)
match = parser.search(str(significados_car[0])
if match.group() == '.n.':
        print ('car es un nombre')
```

```
##################
if wn.synsets('car')[0].pos=='n':
    print('car es un nombre')
##################
if 'n' in str(wn.synsets('car')[0].name):
    print('car es nombre')
# Falla porque si la palabra que se busca incluye la letra "n" , como la palab
ra "knife", "n" también está en el identificador, por ejemplo, "knife.n.01", h
ay dos "n" dentro, pero la primera no significa que la categoría, sino la letr
a que está en la palabra.
```

Paso 3. Exploramos dentro de un synset:¶

- Obtener los lemas del primer synset. Responder a las cuestiones siguientes: ¿qué método(s) se utiliza(n)? ¿cuál es la estructura de datos de la salida? desde el punto de vista lingüístico, ¿qué relación une a estos lemas?
- Obtener la definición del primer synset. ¿De qué tipo de datos es? ¿cómo se obtiene la definición en tipo de datos string?
- Obtener los ejemplos del primer synset. ¿Cuántos ejemplos hay?

```
>>> significados_car[0].lemmas

>>> significados_car[0].lemma_names

>>> significados_car[0].definition

>>> significados_car[0].examples
```

Paso4. Construimos estructuras de datos para almacenar de forma compacta múltiples datos

- Obtener una lista con los lemas de cada significado (synset) de 'car' (ejemplo: [['car', 'auto', automobile', 'machine', 'motorcar'], ["siguiente grupo de lemas"],...] ---> Revisar las instrucciones iterativas (bucles)

Cuando programes:

- Vete poco a poco cada vez una instrucción y de más fácil a más difícil.
- Primero veo qué valores toma la variable de iteración syn,
- segundo, consigo los lemas (tipo Lemma)
- tercero, obtengo los nombres de los lemas,
- y, finalmente, añado los nombres a la lista_sinonimos

Esta técnica de programación se llama divide-y-vencerás. En vez de atacar el problema completo lo divido en subproblemas más sencillos y los resuelvo. Luego compongo la solución completa

```
lista_sinonimos = []
for syn in enumerate(wn.synsets('car')):
    #print(syn)
    #print(wn.synsets('car')[syn[0]].lemmas())
    #print(wn.synsets('car')[syn[0]].lemma_names())
    lista_sinonimos.append(wn.synsets('car')[syn[0]].lemma_names())


lista_sinonimos
```

```
# opcionalmente cuando se trata de crear una lista se puede utilizar el bucle
# en su interior y cuando se ejecuta va agregando los resultados a la lista


[wn.synsets('car')[syn[0]].lemma_names() for syn in enumerate(wn.synsets('car'
))]
#[lemma.name() for lemma in wn.synset('car.n.01').lemmas()]
Resultado final
>>> lista_sinonimos_car = []
>>> for synset in significados_car:
        lista_sinonimos_car.append(synset.lemma_names)
otra posibilidad:
lemas_car = []


for n in range(0,max):
    lemas = significados_car[n].lemmas()
    nombres=[lemas[l].name() for l in range(0, len(lemas))]
    lemas_car.append(nombres)
print(lemas_car)
```

```
[[u'car', u'auto', u'automobile', u'machine', u'motorcar'],
 [u'car', u'railcar', u'railway_car', u'railroad_car'],
 [u'car', u'gondola'],
 [u'car', u'elevator_car'],
 [u'cable_car', u'car']]
```

Paso 5. Crear una función para obtener la lista de sinónimos para cualquier palabra

---> Revisar la definición y uso de funciones

```
def sinonimos(word):
    lista_sinonimos = []
    for (i,synset) in enumerate(wn.synsets(word)):
        lista_sinonimos.append(synset.lemma_names)
    return lista_sinonimos



## Probamos la función
```

```
sinonimos('car')
```

```
[Lemma('car.n.01.car'), Lemma('car.n.01.auto'), Lemma('car.n.01.automobile'),
Lemma('car.n.01.machine'), Lemma('car.n.01.motorcar')]
[Lemma('car.n.02.car'), Lemma('car.n.02.railcar'), Lemma('car.n.02.railway_car
'), Lemma('car.n.02.railroad_car')]
[Lemma('car.n.03.car'), Lemma('car.n.03.gondola')]
[Lemma('car.n.04.car'), Lemma('car.n.04.elevator_car')]
[Lemma('cable_car.n.01.cable_car'), Lemma('cable_car.n.01.car')]

[u'car', u'auto', u'automobile', u'machine', u'motorcar']
[u'car', u'railcar', u'railway_car', u'railroad_car']
[u'car', u'gondola']
[u'car', u'elevator_car']
[u'cable_car', u'car']
```

Paso 6. Crear un diccionario con la descripción de cada synset de una palabra mediante un vector de palabras¶

- Crear un diccionario que almacene por cada identificador de synset (clave) una tupla con las palabas de la definición de cada synset. Por ejemplo, para la palabra 'car': {'car.n.01':['a', 'motor', 'vehicle', 'with', 'four', 'wheels', 'usually', 'propelled', 'by', 'an', 'internal', 'combustion', 'engine'], 'car.n.02': [...], ...} Esta representación de los conceptos mediantes vectores de palabras es un tipo de representación semántica muy útil para comparar lo cercanos (similitud) que pueden ser dos conceptos. ---> Revisar diccionarios

```
# Primero voy a probar cómo se metería manualmente un dato en el diccionario

d = {}

d['car.n.01']=('a', 'motor', 'vehicle', 'with', 'four', 'wheels', 'usually', '
propelled', 'by', 'an', 'internal', 'combustion', 'engine')

d
```

```
{'car.n.01': ('a',
  'motor',
  'vehicle',
  'with',
  'four',
  'wheels',
  'usually',
  'propelled',
  'by',
  'an',
  'internal',
  'combustion',
  'engine')}
```

```
## Luego, para cada significado obtener el identificador y la lista de palabra
s de la glosa si la tiene. Si no tiene glosa usar el vector vacío

## Por ejemplo, el primer identificador sería: id = str(wn.synsets('car')[0].n
ame())

## La definición del primer identificador sería: wn.synset(id).definition()


id = str(wn.synsets('car')[0].name())
```

```
glosa = wn.synset(id).definition()

glosa
```

```
u'a motor vehicle with four wheels; usually propelled by an internal combustio
n engine'
```

```
## Me queda segmentar la definición en palabras (tokenizar), lo haré con split
() aunque no elimina los símbolos de puntuación

glosa.split()
```

```
# Poniendo todo junto en un bucle que itere sobre el número de significados y
vaya añadiendo al diccionario

# cada par id: vector de palabras:

semantica_pal= {}

for synset in wn.synsets(pal):

    id = str(synset.name)

    glosa = wn.synset(id).definition().split()

    semantica_pal[id] = glosa

print(semantica_pal)

---------------
```

```
def crear_diccionario(palabra):

'''toma 1 argumento, la palabra de tipo string'''

      dicc_def_palabra = {}

      sig_palabra = wn.synsets(palabra)

      for synset in sig_palabra:

            clave = synset.name

            definicion = synset.definition.split()

            dicc_def_palabra[clave] = definicion

      return dicc_def_palabra
```

```

```

```
{'car.n.04': [u'where', u'passengers', u'ride', u'up', u'and', u'down'], 'cabl
e_car.n.01': [u'a', u'conveyance', u'for', u'passengers', u'or', u'freight', u
'on', u'a', u'cable', u'railway'], 'car.n.01': [u'a', u'motor', u'vehicle', u'
with', u'four', u'wheels;', u'usually', u'propelled', u'by', u'an', u'internal
', u'combustion', u'engine'], 'car.n.02': [u'a', u'wheeled', u'vehicle', u'ada
pted', u'to', u'the', u'rails', u'of', u'railroad'], 'car.n.03': [u'the', u'co
mpartment', u'that', u'is', u'suspended', u'from', u'an', u'airship', u'and',
u'that', u'carries', u'personnel', u'and', u'the', u'cargo', u'and', u'the', u
'power', u'plant']}
```

Sugerencia: Ahora puedes crear una bonita función que calcule la representación semántica de una palabra basada en wordnet y vectores de palabras. ¿Cuándo dos palabras serán semánticamente similares?

Paso 7. Volcar los resultados en un archivo para poder utilizarlos en otros programas¶

```
### Serialización de objetos complejos (estructuras de datos) con with JSON


El objetivo es almacenar la representación semántica de las palabras en un arc
hivo manteniendo las estructuras de datos (diccionario o tabla) que hemos crea
do y tenemos en memoria. Esto se conoce como serialización. Tiene una ventaja
importante y es que podemos utilizar los datos ya estructurados en procesamien
tos posteriores sin tener que crear las estructuras cada vez.
```

Serialización de objetos complejos (estructuras de datos) con with JSON¶

El objetivo es almacenar la representación semántica de las palabras en un archivo manteniendo las estructuras de datos (diccionario o tabla) que hemos creado y tenemos en memoria. Esto se conoce como serialización. Tiene una ventaja importante y es que podemos utilizar los datos ya estructurados en procesamientos posteriores sin tener que crear las estructuras cada vez.

```
import json

# cuidado aquí debes poner la ruta de la carpeta donde se creará el archivo js
on

root = 'C:/Users/..../PLN/practicas2015_16/1_revision_programacion_wn/'


with open('datos.json', 'w') as g:

    json.dump(semantica_pal,g)
```

Fin de la revisión. Si algo no te ha salido repítelo dentro de un día o dos, así hasta que te salga. No cejes en tu empeño de aprender. "No se hacen las cosas simplemente por placer o por que a uno le guste, sino por llegar a lo que uno se ha propuesto" (Enrique Rojas)