PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

FACULTAD DE LETRAS

Introducción a la lingüística computacional

César Antonio Aguilar

Facultad de Lenguas y Letras

24/08/2017

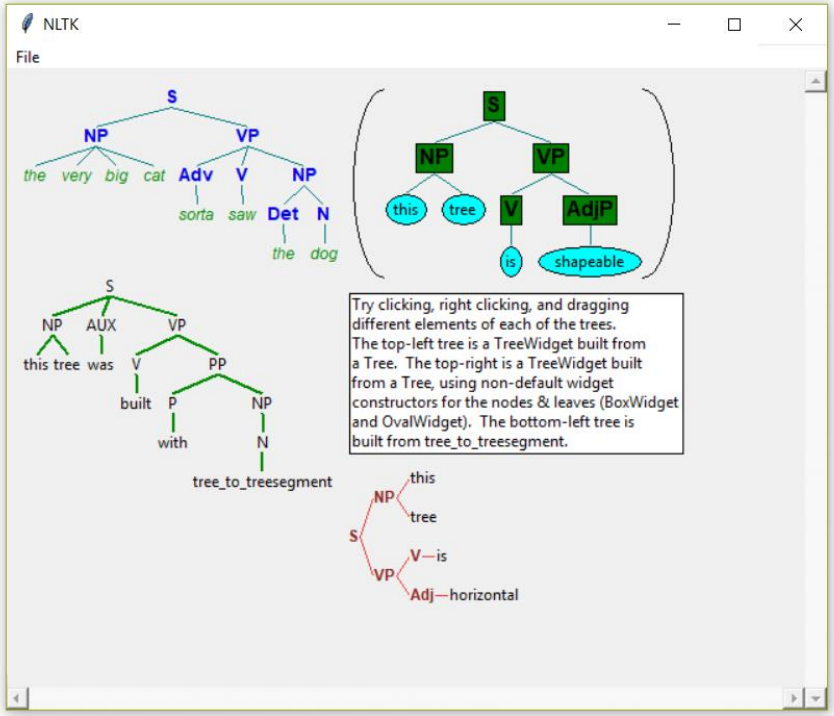
Cesar.Aguilar72@gmail.com

Explorando NLTK (1)

En esta sesión, vamos a ver algunos de los recursos básicos que tiene NLTK, así como avanzar en cuestiones de programación relacionadas con el uso de expresiones regulares (Regexp).

```
In [1]: import nltk

In [2]: nltk.draw.tree.demo()
```



Explorando NLTK (2)

A grandes rasgos, NLTK es una colección de herramientas que nos permite hacer varias tareas: desde procesar uno o varios documentos para fines sencillos (contar palabras, agrupar tokens, observar distribuciones de frecuencias, etc.), hasta cuestiones como análisis sintáctico, desambigüación de frases y/u oraciones y tareas de extracción de información.

Nota: aquí tienen la liga a un video en donde Edward Loper, Ewan Klein y Steven Bird, los inventores de NLTK, platican durante una hora (aprox.) qué es NLTK y qué se puede hacer con él.

www.youtube.com/watch?v=keXW_5-llD0

NLTK Book (1)

Hagamos una pequeña exploración por nuestra cuenta: de momento concentrémonos en dos módulos de NLTK: uno denominado *NLTK Book*, y otro que colecta todas las *demos* que varios desarrolladores han elaborado e integrado a NLTK.

Empecemos por el módulo *Book*. Escriban en el intérprete de Python la siguiente instrucción:

```
from nltk.book import *
```

NLTK Book (2)

Si aplicamos esta instrucción, nos van a salir 9 archivos denominados *texts* (1-9), los cuales son documentos tales como novelas, guiones de cine, fragmentos de prensa escrita, etc.:

```
In [3]: from nltk.book import *
*** 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
```

NLTK Book (3)

Con estos textos, podemos hacer algunas tareas. P.e., usemos el comando **concordance**, realizamos una búsqueda sencilla de concordancias:
concordancias:

```
text1.concordance("monstrous")
```

Vean los resultados en la lámina siguiente:

NLTK Book (4)

Otra tarea sencilla que podemos realizar es reconocer qué palabras muestran un rango similar de aparición dentro de los contextos de nuestras concordancias:

```
>>> text6.similar("Camelot")
Building word-context index...
and ni what brave that sir many why arimathea confuse fight leave
course someone good have nothing peril come war
>>>
```

NLTK Book (4)

```
Displaying 11 of 11 matches:
ong the former , one was of a most monstrous size . ... This came towards us ,
ON OF THE PSALMS . " Touching that monstrous bulk of the whale or ork we have r
ll over with a heathenish array of monstrous clubs and spears . Some were thick
d as you gazed , and wondered what monstrous cannibal and savage could ever hav
that has survived the flood ; most monstrous and most mountainous ! That Himmal
they might scout at Moby Dick as a monstrous fable , or still worse and more de
th of Radney .'" CHAPTER 55 Of the Monstrous Pictures of Whales . I shall ere l
ing Scenes . In connexion with the monstrous pictures of whales , I am strongly
ere to enter upon those still more monstrous stories of them which are to be fo
ght have been rummaged out of this monstrous cabinet there is no telling . But
of Whale - Bones ; for Whales of a monstrous size are oftentimes cast up dead u
```

NLTK Book (5)

Ahora, veamos la siguiente función, denominada **similar**:

text1.similar("monstrous")

Lo que esta función nos permite es reconocer palabras que son empleadas en contextos similares, p. e., *the* ___ *pictures* o *a* ___ *size*. Veamos:

```
In [8]: text1.similar("monstrous")
singular vexatious horrible subtly uncommon untoward determined
maddens contemptible tyrannical puzzled christian mystifying candid
careful gamesome lazy few perilous wise
```

NLTK Book (6)

Podemos acotar el espacio de búsqueda para estos contextos, usando pares de palabras. Para ello, ocupamos la instrucción: **common_contexts**:

```
In [11]: text1.common_contexts(["monstrous", "uncommon"])
most_and

In [12]: text1.common_contexts(["monstrous", "horrible"])
most_and
```

P.e., usando la palabra “monstrous”, y combinándolas con “horrible” y “uncommon”, obtenemos algunas palabras asociadas que aparecen en el contexto de las concordancias.

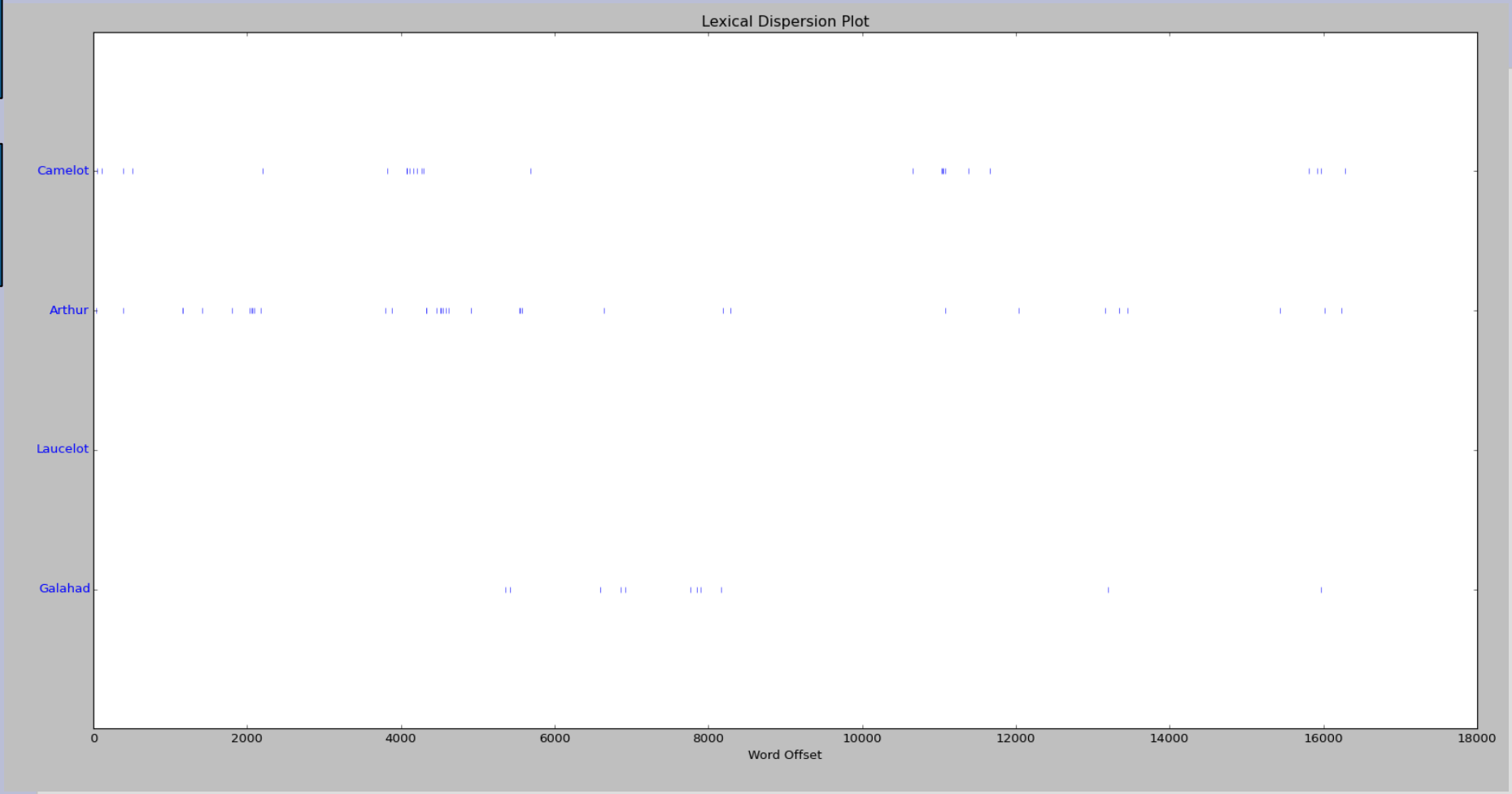
NLTK Book (7)

Este módulo ya nos permite hacer algunos análisis de frecuencias de uso entre palabras. P. e., con el comando **dispersion_plot**, podemos crear un grafo de dispersión. Primero, escribimos la siguiente instrucción:

```
In [13]: text6.dispersion_plot(["Camelot", "Arthur", "Laucelot", "Galahad"])
```

Y el resultado lo podemos ver en la siguiente lámina:

NLTK Book (8)


Lexical Dispersion Plot
Camelot
Arthur
Laucelot
Galahad
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
Word Offset

Importando archivos (1)

Aprovechemos un poco estas herramientas para ver si podemos obtener algo con textos nuevos. De momento, trabajaremos con documentos en inglés.

Para importar un archivo a nuestro intérprete de Python, requerimos utilizar la instrucción **open**, y escribir la ruta de acceso a nuestro archivo del siguiente modo:

```
Cadena_Ariel01 = open('C://Anaconda3/Corpus_Pruebas/ariel01.txt',
'rU').read()
```

Importando archivos (2)

Ahora, corroboremos que hemos asociado este archivo a nuestro nombre de variable. Para esto, usemos las funciones **type** y **len**. ¿Qué obtenemos? Veamos:

```
In [15]: type(Cadena_Ariel01)
Out[15]: str

In [16]: len(Cadena_Ariel01)
Out[16]: 144249
```

Importando archivos (3)

Con la instrucción anterior lo que hicimos fue “cargar” un archivo, el cual fue transformado en una cadena de caracteres tras el uso de comando anterior.

Esta cadena cuenta con una propiedad: que está “cruda” (*raw*), indicado por la letra *r*, ya que la hemos generado a partir de nuestro archivo. Por otro lado, señalamos que esta cadena también puede tener caracteres Unicode, representados con la letra *U*.

Así, lo que obtenemos es una cadena. Ahora, lo que requerimos en convertirla en un **conjunto de palabras**. Para eso, ocupamos las siguientes instrucciones:

```
import re

Lista_Ariel01 = re.split(r'_espacio en blanco_', Cadena_Ariel01)
```

Importando archivos (4)

¿Qué fue lo que hicimos con la instrucción anterior? Convertir nuestra cadena en una **lista** (o en un conjunto). Como objetos, las listas tienen permiten empezar a hacer algunos tratamiento particulares al PLN, como por ejemplo la asignación de etiquetas morfosintácticas. Por lo pronto, lo que tenemos es un grupo de diferentes objetos:

```
In [22]: Lista_Ariel01 = re.split(r' ', Cadena_Ariel01)

In [23]: type(Lista_Ariel01)
Out[23]: list

In [24]: len(Lista_Ariel01)
Out[24]: 25004
```

Importando archivos (5)

La instrucción anterior opera del siguiente modo:

1. Al inicio de nuestro código, debemos importar un módulo llamado **re**, el cual nos sirve para realizar operaciones con expresiones regulares.

2. Una vez importando este módulo, vamos a ocupar la función **split**, que literalmente nos ayuda a *rebanar* los elementos que conforman nuestra cadena, de tal forma que obtenemos un listado de objetos.

3. Los objetos que conforman nuestra lista son palabras, números y signos de puntuación. Obviamente, no estamos considerando aquí tablas, gráficos, esquemas, etc.

Importando archivos (6)

Hagamos una mejor obtención de palabras. La variable **W**, que significa *word*, ya nos permite segmentar únicamente aquellas cosas que podemos ver como formas de palabras:

```
Lista_Ariel02 = re.split(r'\W+', Cadena_Ariel01)
```

Así, nuestra instrucción se traduce como: “a partir del módulo Regular Expresions, aplica la función *split*, en concreto de un texto crudo (*raw*, *r*), secciona aquellos objetos que tengan el atributo *word*. Aplica esta función al objeto Cadena_Ariel02”. Veamos el resultado:

Importando archivos (6)

```
In [25]: Lista_Ariel02 = re.split(r'\W+', Cadena_Ariel01)

In [26]: type(Lista_Ariel02)
Out[26]: list

In [27]: len(Lista_Ariel02)
Out[27]: 21606
```

Importando archivos (7)

¿Qué tanta diferencia hay entre estos dos métodos? Comparemos usando rangos, en este caso las primeras 50 palabras. Nuestros resultados:

```
In [28]: Rango_Ariel01 = Lista_Ariel01[0:50]

In [29]: print(Rango_Ariel01)
['THE', 'DEVELOPMENT', 'OF', 'PERSON', 'AGREEMENT', 'MARKERS:\nFROM', 'PRONOUNS', 'TO', 'HIGHER', 'ACCESSIBILITY',
'MARKERS*\n\nMira', 'Ariel', '(1999)\nTel-Aviv', 'University\n\n\n\n1', '.', '', 'From', 'free', 'pronouns', 'to',
'verbal', 'agreement:', 'Introduction', '\n\nGrammatical', 'items', 'often', 'begin', 'their', 'linguistic', 'life',
'as', 'regular', 'lexical', 'items', '(Meillet', '1912)', '.', '', 'Grammaticization', 'is', 'said', 'to', 'have',
'occurred', 'when', 'the', 'position', 'of', 'such', 'lexemes']

In [30]: Rango_Ariel02 = Lista_Ariel02[0:50]

In [31]: print(Rango_Ariel02)
['THE', 'DEVELOPMENT', 'OF', 'PERSON', 'AGREEMENT', 'MARKERS', 'FROM', 'PRONOUNS', 'TO', 'HIGHER', 'ACCESSIBILITY',
'MARKERS', 'Mira', 'Ariel', '1999', 'Tel', 'Aviv', 'University', '1', 'From', 'free', 'pronouns', 'to', 'verbal',
'agreement', 'Introduction', 'Grammatical', 'items', 'often', 'begin', 'their', 'linguistic', 'life', 'as', 'regular',
'lexical', 'items', 'Meillet', '1912', 'Grammaticization', 'is', 'said', 'to', 'have', 'occurred', 'when', 'the',
'position', 'of', 'such']
```

Nota: expresiones regulares (1)

Python, como muchos lenguajes de programación, cuenta con métodos y colecciones de expresiones regulares. En este caso, se agrupan en el módulo **re**. Veamos algunas de las más conocidas:

Operaciones con conjuntos

Expresión regular	Significado
/[A-Z]/	"Busca todos los caracteres que sean mayúsculas"
/[a-z]/	"Busca todos los caracteres que sean minúsculas"
/[0-9]/	"Busca todos los números de 0 a 9"

Negaciones

Expresión regular	Significado
[^A-Z]	"Busca todos los caracteres menos mayúsculas"
[^S-s]	"Busca cualquier carácter menos 'S' o 's'"
[^\.]	"No es un párrafo"
[e^]	"Cualquiera que sea 'e' o 'vacío'"
a ^ b	"Busca el patrón 'a_b'"

Nota: expresiones regulares (2)

Operaciones con cadenas

Expresión regular	Significado
/ ? (Disyunción)/	“’Computadora’ OR ‘Ordenador’”
* (Estrella de Klenne)	“’Busca desde 0 hasta infinito’”
+ (Suma o adición)	“Busca desde 1 hasta infinito”
/ Constan. / (Comodín)	“’Constante OR ‘Constancia’”
/ ^ [A-Z] /	“’El Colegio de México’ OR ‘Colegio de México’”
/ ^ [^A-Z] /	“’¿Verdad?’ OR ‘Really?’”
/ \. $/	“’Aquí termina el párrafo.’”
/ . $/	“’?’, ‘!’”
/ \b on \b /	“’Internacional’ → ‘Internacional’”
/ casa | caza /	“Ir a la caza de una casa’”

Importando archivos (8)

Ahora, ¿qué podemos hacer con este procesamiento? Supongamos que una tarea tan tediosa como “limpiar un corpus”, podemos automatizarla usando expresiones regulares, incluso desde la importación de nuestro archivo. Veamos la siguiente lista de expresiones regulares:

```
>>> pattern = r'''(?x)    # set flag to allow verbose regexps
    ([A-Z]\.)+        # abbreviations, e.g. U.S.A
    | \w+(-\w+)*        # words with optional internal hyphens
    | \$?\d+(\.\d+)?%?  # currency and percentages, e.g. $12.40, 82%
    | \.\.\.            # ellipsis
    | [][.,;"'?():-_`]  # these are separate tokens
    '''
```

Gracias por su atención

Blog del curso:
http://cesaraguilar.weebly.com/curso-de-procesamiento-del-lenguaje-natural.html