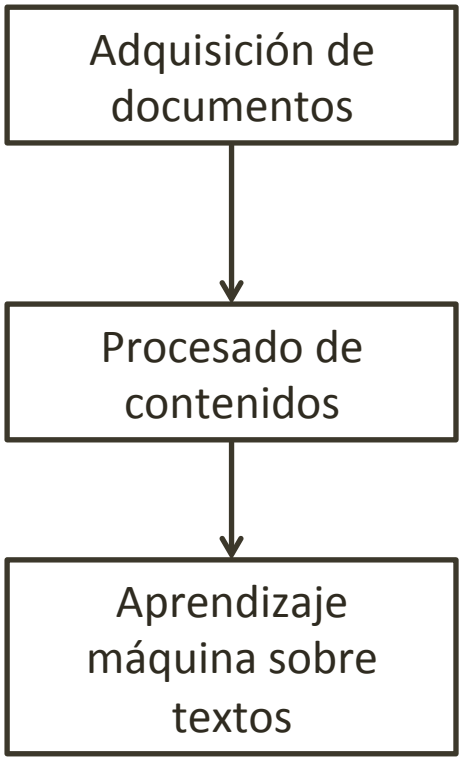
Natural Language Processing en Python

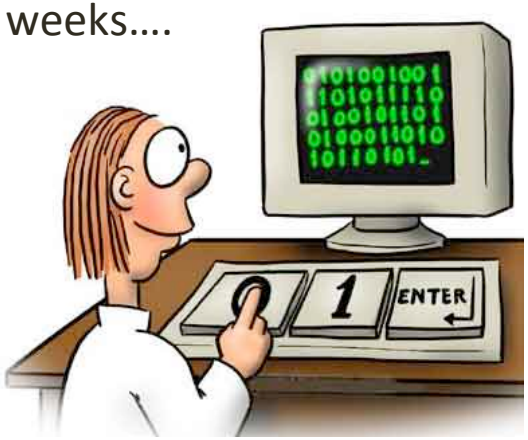
Vanessa Gómez Verdejo
Manel Martínez Ramón

Objetivos de esta sesión



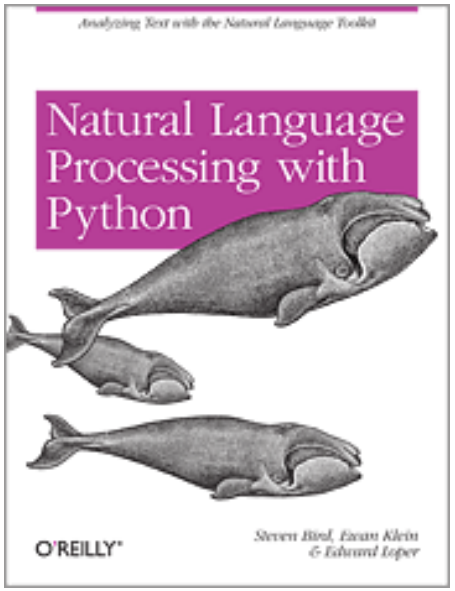
NLTK (Natural Language Toolkit)

In 2 weeks….

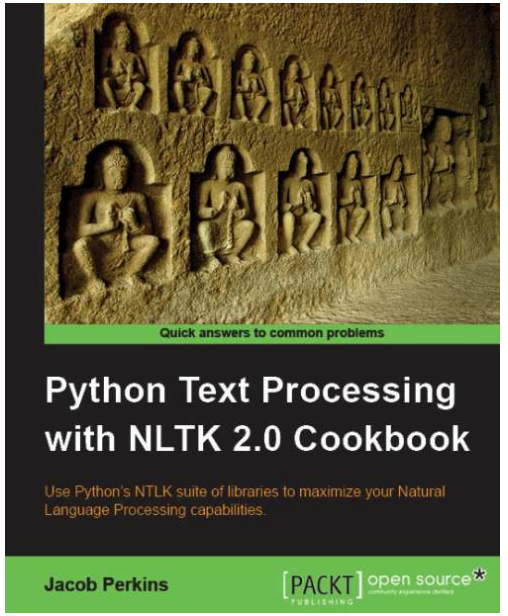


you'll be NLTK geeks!

Referencias básicas



nltk.org/book



Empezando a manejar NLTK

Parte 1

Instalación y arranque

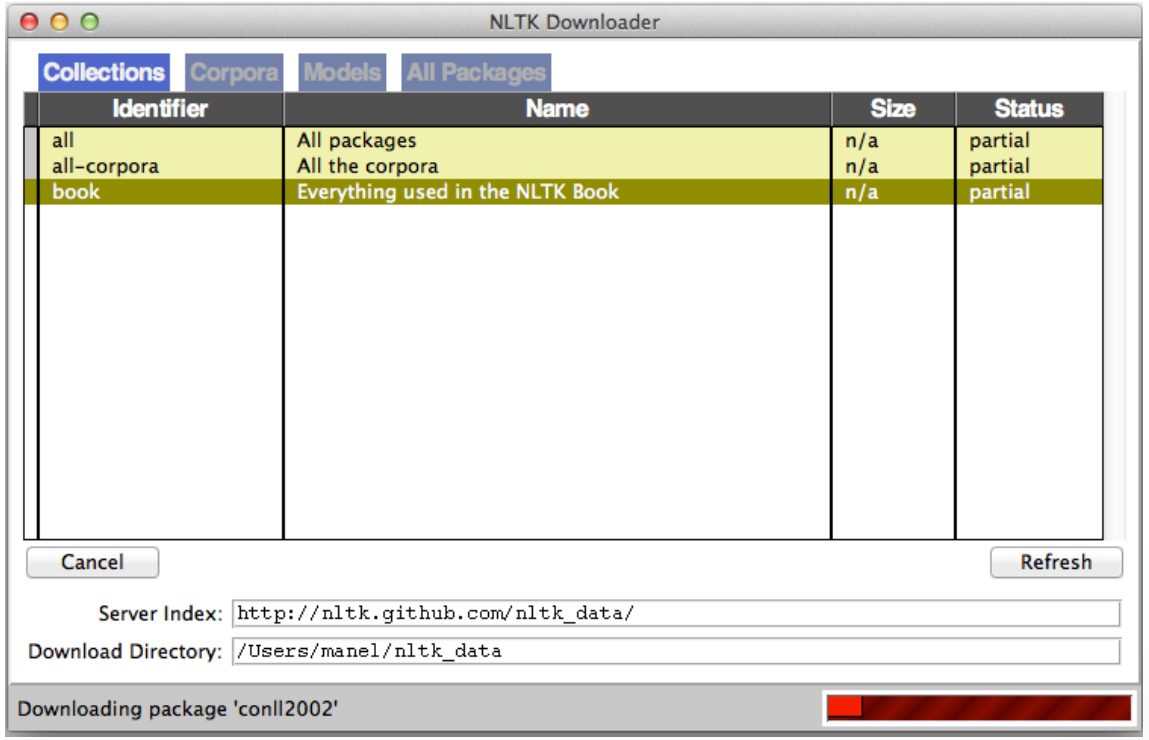
- Se supone que el usuario tiene instalado el Python, con las librerías:
  - NLTK
  - NLTK-Data
  - NumPy, Scipy
  - Pylab
- Por sencillez se usará el ipython notebook en las prácticas
- Puesta en marcha: arrancar python y ejecutar

```
>> ipython notebook
import numpy
import pylab
import nltk
from __future__ import division
```

Textos disponibles en NLTK

- `nltk.download()` `#repositorio de paquetes`

Se abre la ventana de importación, donde se deben seleccionar los datos.



Carga de corpora

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

- 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

La clase text

- Los anteriores textos pertenecen a la clase “text” de la librería nltk, e incorporan diferentes funciones.
- Algunas funciones útiles para análisis semántico
- text2.concordance(‘monstrous’)
  - Devuelve las líneas del texto “text1” en las que aparece “monstrous”
- text2.similar(‘monstrous’)
  - Devuelve palabras usadas de forma similar en el texto.
- text2.common_contexts([‘monstrous’,’very’])
  - Encuentra contextos compartidos por dos o más palabras
  - Los “_” del resultado denotan dónde van las palabras.
- text2.tokens
  - Crea una lista con los tokens del texto

Funciones útiles de Python

- len(text1)
- set(text1)
- sorted(set(text1))

- Obtenemos el vocabulario (por orden alfabético) de un texto

```
words=sorted(set(text1))
```

- Calculamos la frecuencia de aparición de una palabra

```
def percentage(count,total):
    return 100*count/total
```

- Trabajando con listas recursivamente

```
[percentage(text.count(word),len(text1)) for word in words]
[word for word in words if percentage(text1.count(word),len(text1))>4]
```

El tipo str de Python

- s.startswith(t) test if s starts with t
- s.endswith(t) test if s ends with t
- t in s test if t is contained inside s
- s.islower() test if all cased characters in s are lowercase
- s.isupper() test if all cased characters in s are uppercase#
- s.isalpha() test if all characters in s are alphabetic
- s.isalnum() test if all characters in s are alphanumeric
- s.isdigit() test if all characters in s are digits
- s.istitle() test if s is titlecased (all words in s have have initial capitals)

- [w for w in set(text1) if w.endswith('ableness')]

El tipo str de Python

- s.find(t) index of first instance of string t inside s (-1 if not found)
- s.rfind(t) index of last instance of string t inside s (-1 if not found)
- s.index(t) like s.find(t) except it raises ValueError if not found
- s.rindex(t) like s.rfind(t) except it raises ValueError if not found
- s.join(text) combine the words of the text into a string using s as the glue
- s.split(t) split s into a list wherever a t is found (whitespace by default)
- s.splitlines() split s into a list of strings, one per line
- s.lower() a lowercased version of the string s
- s.upper() an uppercased version of the string s
- s.title() a titlecased version of the string s
- s.strip() a copy of s without leading or trailing whitespace
- s.replace(t, u) replace instances of t with u inside s
- s.startswith(t) test if s starts with t
- s.endswith(t) test if s ends with t
- t in s test if t is contained inside s
- s.islower() test if all cased characters in s are lowercase
- s.isupper() test if all cased characters in s are uppercase
- s.isalpha() test if all characters in s are alphabetic
- s.isalnum() test if all characters in s are alphanumeric
- s.isdigit() test if all characters in s are digits
- s.istitle() test if s is titlecased (all words in s have have initial capitals)

Expresiones útiles

- EJERCICIO 1
- Homogeneizar el vocabulario de un texto poniendo todas las palabras en minúsculas

- EJERCICIO 2
- Eliminar los signos de puntuación de un texto

La función FreqDist

fdist=FreqDist(text2)

fdist es una función que genera una variable de la clase nltk.probability.freqdist.

Incorpora diferentes funciones que permiten generar el vocabulario del texto y la frecuencia de aparación de cada elemento (en orden decreciente). Incluye, entre otras, las siguientes funciones:

fdist.samples()	Genera una lista con el vocabulario
fdist.values()	Devuelve el nº de veces que aparece cada palabra
fdist.N()	Nº total de palabras en el texto
fdist.freq(*word*)	Devuelve la frecuencia de aparación de *word*
fdist.inc(palabra)	Incrementa la cuenta para una palabra
fdist.max()	Palabra con máxima frecuencia
fdist.tabulate()	Tabula la distribución de frecuencia
fdist.plot()	Representación gráfica

My ~~first~~ second bag of words

EJERCICIO 1:

Utilizando FreqDist generar el bag-of-words de text1 y visualizarlo con .tabulate()

EJERCICIO 2:

Utilizando FreqDist, genere una lista de tuplas conteniendo las palabras de más de 16 caracteres y su frecuencia de aparición. Por ejemplo:

['companionableness', 7.063344069616319e-06], ['disinterestedness', 7.063344069616319e-06], ['disqualifications', 7.063344069616319e-06],....

Corpora de interés

- Cómo importar un corpus
  - Vemos sus textos

    ```
    nltk.corpus.gutenberg.fileids()
    ['austen-emma.txt',  'austen-persuasion.txt', ...
    ```

  - Seleccionamos uno de ellos

    ```
    emma = nltk.corpus.gutenberg.words('austen-emma.txt')
    ```

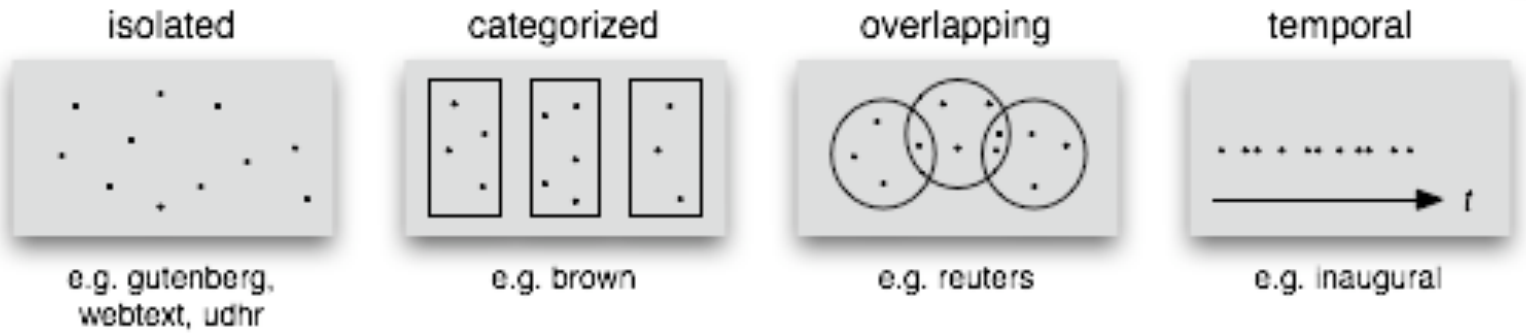
  - O, directamente

    ```
    from nltk.corpus import gutenberg
    emma = gutenberg.words('austen-emma.txt')
    ```

Funciones del corpus

fileids()	the files of the corpus
fileids([categories])	the files of the corpus corresponding to these categories
categories()	the categories of the corpus
categories([fileids])	the categories of the corpus corresponding to these files
raw()	the raw content of the corpus
raw(fileids=[f1,f2,f3])	the raw content of the specified files
raw(categories=[c1,c2])	the raw content of the specified categories
words()	the words of the whole corpus
words(fileids=[f1,f2,f3])	the words of the specified fileids
words(categories=[c1,c2])	the words of the specified categories
sents()	the sentences of the whole corpus
sents(fileids=[f1,f2,f3])	the sentences of the specified fileids
sents(categories=[c1,c2])	the sentences of the specified categories
abspath(fileid)	the location of the given file on disk
encoding(fileid)	the encoding of the file (if known)
open(fileid)	open a stream for reading the given corpus file
root()	the path to the root of locally installed corpus
readme()	the contents of the README file of the corpus

Corpora etiquetados



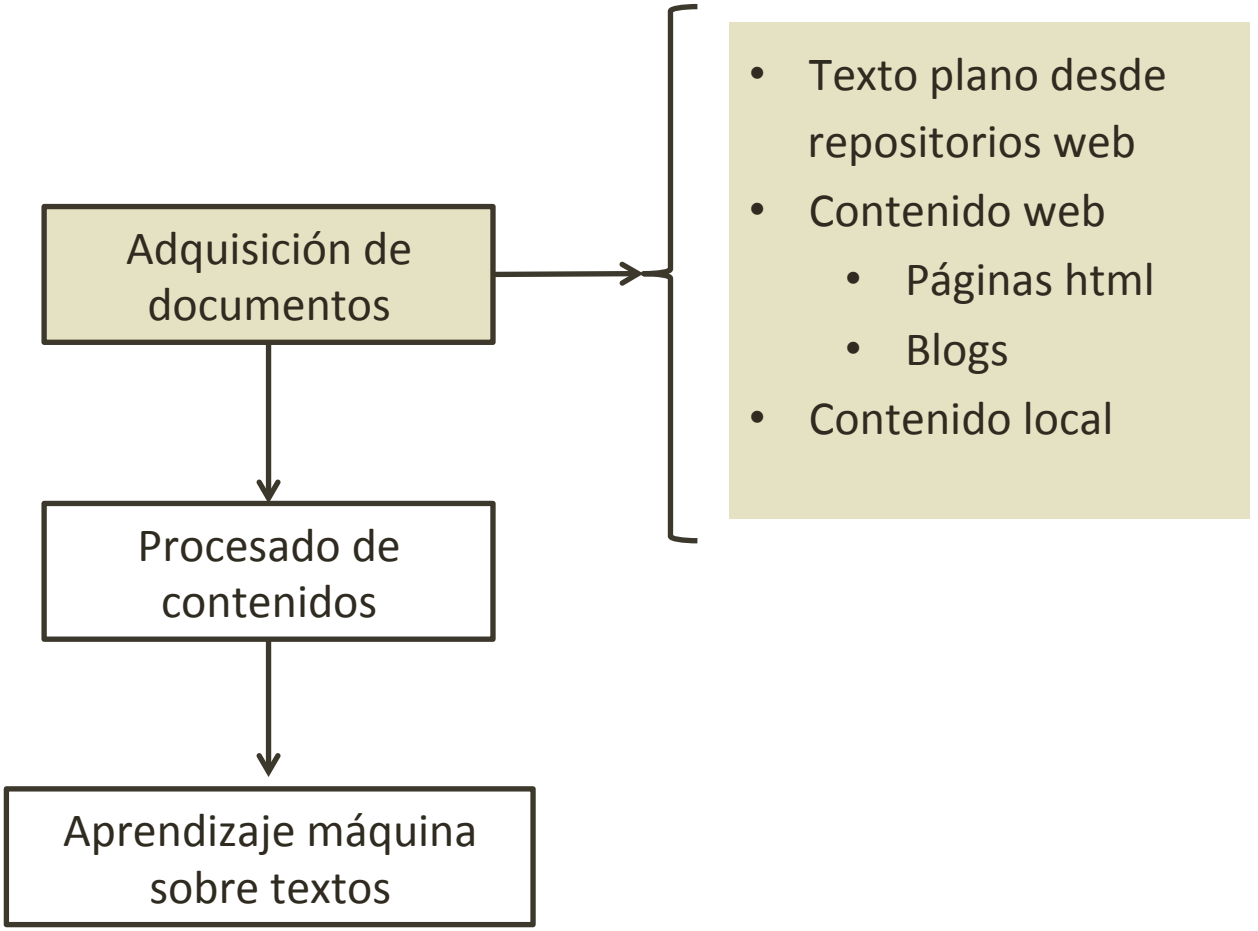
EJEMPLO:

- from nltk.corpus import brown
- brown.categories()
- [[fileid, brown.categories(fileid)] for fileid in brown.fileids()]

Adquisición de documentos

Parte 2

Objetivos


Adquisición de documentos
Procesado de contenidos
Aprendizaje máquina sobre textos
• Texto plano desde repositorios web
• Contenido web
• Páginas html
• Blogs
• Contenido local

Importación de texto plano

- Se puede importar cualquier texto plano. La descarga desde un repositorio web puede hacerse así:

```
from urllib import urlopen
url = "http://www.gutenberg.org/files/2554/2554.txt"
raw = urlopen(url).read()
```

- En la variable “raw” habrá texto crudo. Antes de proceder a su procesado hay que formatearlo (tokenization).
- El formato consiste simplemente en convertirlo en una lista de elementos (palabras, frases, bigramas, por ejemplo).

```
tokens = nltk.word_tokenize(raw)
```

Importación de páginas web

- Particularmente interesante es la posibilidad de leer cualquier página web

```
url = "http://news.bbc.co.uk/2/hi/health/2284783.stm"
texto = urlopen(url).read()
```

- Este obviamente es un artículo de BBC News. En efecto, si hacemos:

```
texto[715:800]
```

- obtenemos cualquier disparate. ¿Cuál en este caso?

Importación de páginas web

- Las etiquetas HTML se pueden eliminar automáticamente antes de la “tokenización” (pero las frases absurdas no):

```
raw = nltk.clean_html(html)
tokens = nltk.word_tokenize(raw)
tokens[:10]
```

['BBC', 'NEWS', '|', 'Health', '|', 'Blondes', "'to", 'die', 'out', 'in']

Importación de blogs

- Se puede importar RSS

```
import feedparser
llog = feedparser.parse("http://languagelog.ldc.upenn.edu/nll/?feed=atom")
```

- Título

```
post = llog.entries[2]
post.title
```

u'NPR: oyez.org finishes Supreme Court oral arguments project'

- Contenido

```
content = post.content[0].value
```

Importación de contenido local

- Datos locales

```
path = nltk.data.find('corpora/gutenberg/melville-moby_dick.txt')   #Busca un texto
raw = open(path, 'rU').read()                                       #Carga el texto
tokens = nltk.word_tokenize(raw)                                    #Formato
raw[1:29]
'Moby Dick by Herman Melville'
```

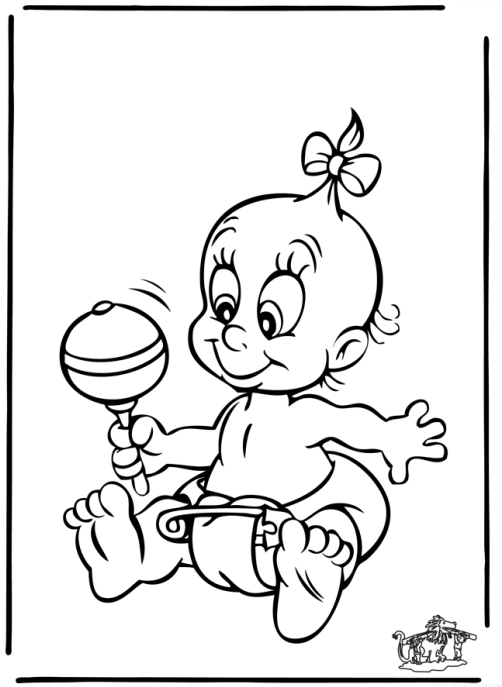
- Desde el teclado (no funciona en iPython)

```
s = raw_input("Enter some text: ")
```

Un vocabulario muy simple

- Construcción de un vocabulario (véanse operaciones con strings)

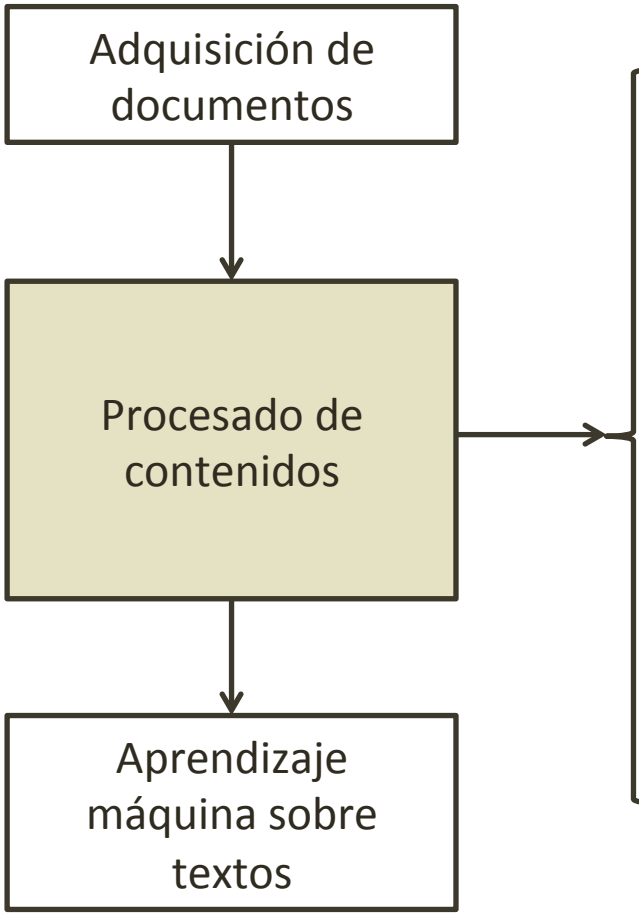
```
words = [w.lower() for w in tokens]
            # eliminamos mayúsculas
vocab = sorted(set(words))
len(vocab)
21317

words = [w.lower() for w in tokens if w.isalpha()]
            # Nos quedamos solo con el texto
vocab = sorted(set(words))
len(vocab)
15940
```



Procesado de contenido

Parte 3

Objetivos



- Tokenización
- Extracción de raíces: lematización
- Eliminación de stopwords
- Corrección de texto
  - “Spell-checker”
  - Expresiones regulares
  - Caracteres repetidos
- Sinónimos

Tokenización

- Paso de texto a frases

```
from nltk.tokenize import sent_tokenize
frase='Trabajo básicamente en el apartado de la comedia. Me gustaría estar en Diseño de Programación, pero por desgracia aprobé el bachillerato.'
sent_tokenize(frase)
```

```
['Trabajo b\xc3\xa1sicamente en el apartado de la comedia.', 'Me gustar \xc3\xada estar en Dise\xc3\xb1o de Programaci\xc3\xb3n, pero por desgracia aprob\xc3\xa9 el bachillerato.']
```

Utiliza una instancia de PunktSentenceTokenizer, que funciona para una gran variedad de idiomas.

Si se deben formatear muchas frases en inglés, es más eficiente cargar el módulo correspondiente

```
import nltk.data
tokenizer=nltk.data.load('tokenizers/punkt/english.pickle')
tokenizer.tokenize(frase)
```

Tokenización

- Otros idiomas.

  Para formatear texto en frases en otros idiomas cargando previamente la librería, hay que especificarlo:

  tokenizer=nltk.data.load('tokenizers/punkt/spanish.pickle')

- Paso de texto a palabras

  from nltk.tokenize import word_tokenize

  word_tokenize('Hola, mundo.')

  ['Hola', ',' , 'mundo' , '.']

  word_tokenize no elimina la puntuación y puede separar contracciones (en castellano no hay).

Tokenización

- Definición de criterios de separación usando RegexpTokenizer

  from nltk.tokenize import regexp_tokenize

  regexp_tokenize("Can't is a contraction.","[\w']+")

  ["Can't", 'is', 'a', 'contraction']

- Ejemplo: separador por espacios

  regexp_tokenize("Can't is a contraction.","[\S']+")

  ["Can't", 'is', 'a', 'contraction.']

  Nótese que esta vez ha incluido el punto.

  Véase la sección de expresiones regulares para más detalles.

Librería re (expresiones regulares)

.	Wildcard
^abc	Encuentra un patrón abc al inicio de una cadena
abc$	Encuentra abc al final de una cadena
[abc]	Encuentra uno de un conjunto de caracteres
[A-Z0-9]	Encuentra uno de un rango de caracteres
ed|ing|s	Encuentra una de las cadenas especificadas (disjunction)
*	Cero o más de los ítems previos. Ej.: a*, [a-z]* (Kleene Closure)
+	Uno o más de los ítems previos, Ej.: a+, [a-z]+
?	Opcional, ej.: a?, [a-z]?
{n}	Exactamente n repeticiones
{n,}	Al menos n repeticiones
{,n}	No más de n repeticiones
{m,n}	Al menos m y no más de m
a(b|c)+	Paréntesis que indica el objeto de las operaciones

Librería re (expresiones regulares)

Símbolo	**Función**
\b	Word boundary (zero width)
\d	Cualquier decimal (equivalente a [0-9])
\D	Cualquier no decimal (equivalente a [^0-9])
\s	Cualquier espacio (equivalente a [\t\n\r\f\v]
\S	Cualquier no espacio (equivalente a [^ \t\n\r\f\v])
\w	Cualquier alfanumérico (equivalente a [a-zA-Z0-9_])
\W	Cualquier no alfanumérico (equivalente a [^a-zA-Z0-9_])
\t	Tabulador
\n	Nueva línea
\r	Retorno de carro
\f	Form feed (página nueva)
\v	Tabulador vertical

Librería re (expresiones regulares)

- Ejemplo: Palabras que terminan en “ed” en inglés

```
import re          # Importa la la librería
wordlist = [w for w in nltk.corpus.words.words('en') if
w.islower()]     # Cargamos el voabulario inglés

words=[w for w in wordlist if re.search('ed$', w)] #Obtiene las
que terminan en ‘ed’
print words[:10]
```

['abaissed', 'abandoned', 'abased', 'abashed', 'abatised', 'abed', 'aborted', 'abridged', 'abscessed', 'absconded']

Librería re (expresiones regulares)

- Ejemplo: Palabras que contienen ..j..t.. donde los '.' son wildcards

```
words=[w for w in wordlist if re.search('^..j..t..$', w)]
words[:10]
```

['majestic', 'majestic', 'majestic', 'objected', 'majestic', 'majestic', 'abjectly', 'majestic','dejected']

- Nótense las palabras repetidas, porque hemos buscado en el texto completo. Podemos buscar en el vocabulario anteriormente construido

```
words=[w for w in vocab if re.search('^..j..t..$', w)]
words[:10]
```

['abjectly', 'abjectus', 'dejected', 'majestic', 'objected']

Librería re (expresiones regulares)

- Otros ejemplos

```
print [w for w in wordlist if re.search('^m+e+$', w)]
print [w for w in wordlist if re.search('^m*e+$', w)]
print [w for w in wordlist if re.search('^[me]+$', w)]
```

```
['me']
['e', 'me']
['e', 'em', 'eme', 'm', 'me', 'mem']
```

Librería re (expresiones regulares)

- División de texto: re.split(expre,texto)

raw ="'*When I'M a Duchess,' she said to herself, (not in a very hopeful tone though), 'I won't have any pepper in my kitchen AT ALL. Soup does very well without--Maybe it's always pepper that makes people hot-tempered,'...*”

```
re.split(r' ', raw)          #Dividimos cuando hay espacios
re.split(r'[ \t\n]+', raw) #Dividimos cuando hay espacios,
                             #tabulaciones o retornos de carro
re.split(r'\W+', raw)      #Dividimos  cuando aparece cualquier
                             #carácter que NO es alfanumérico
```

Librería re (expresiones regulares)

- Busqueda de expresiones regulares: re.findall(expre,w)

```
word = 'supercafrelisticexpialidocious'
len(re.findall(r'[aeiou]', word))
```

16 o 14?

```
re.findall(r'\w+|\S\w*', raw)
re.findall(r"\w+(?:[-']\w+)*|'|[-.(]+|\S\w*", raw)
```

- Eliminación de sufijos:

```
stem, suffix = re.findall(r'(^.*)(ing|ly|ed|ious|ies|ive|es|s|ment)$',
'processing')[0]
```

- Nótense
  - el uso de los paréntesis r'(…)(…)
  - la asignación a,b = c,d;
  - el uso de r'
  - [0] significa que se accede al primer elemento de la lista devuelta

Extractor de raíces simple

- Ejercicio: Definir una función que nos devuelva la raíz de las palabras, es decir, que elimine los sufijos ing, ly, ed, ious, ies, ive, es, s, ment

Extractores de raíces en NLTK

EXTRACTORES PORTER Y LANCASTER

```
raw = ”DENNIS: Listen, strange women lying in ponds distributing swords
... is no basis for a system of government.  Supreme executive power derives from
... a mandate from the masses, not from some farcical aquatic ceremony.”

tokens = nltk.word_tokenize(raw)

porter = nltk.PorterStemmer()
[porter.stem(t) for t in tokens]
['DENNI', ':', 'Listen', ',', 'strang', 'women', 'lie', 'in', 'pond', ....

lancaster = nltk.LancasterStemmer()
[lancaster.stem(t) for t in tokens]
['den', ':', 'list', ',', 'strange', 'wom', 'lying', 'in', 'pond', 'distribut', ...
```

Extractores de raíces en nltk

EXTRACTOR SNOWBALL

Funciona con 15 lenguas: SnowballStemmer.languages devuelve la lista.

```
from nltk.stem import SnowballStemmer
spanish_stemmer=SnowballStemmer("spanish")
texto='Hola estoy disfrutando de este curso mientras me estoy durmiendo'

tokens = nltk.word_tokenize(texto)
[spanish_stemmer.stem(t) for t in tokens]
```

Lematización

- Un lematizador extrae la auténtica raíz de una palabra a partir de un diccionario.
- NLTK utiliza WordNet, una base de datos léxica en inglés, a través de un interfaz.
- Tiene una versión en castellano (http://grial.uab.es/sensem/download/).

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
[lemmatizer.lemmatize(t) for t in tokens]
```

El resultado ya no contiene las palabras con los sufijos extraídos sino que contiene la raíz o lema de las palabras.

Con Lancaster:

- 'execut', 'power', 'deriv'

Con lematizador:

- 'executive', 'power', 'derives',

Lematización

- El lematizador busca un lema para la palabra, y la devuelve tal cual si no lo halla.

  lemmatizer.lemmatize('findings')

  finding

- Si se quieren encontrar los verbos, hay que especificarlo

  lemmatizer.lemmatize('findings',pos='v')

  find

Reconoce adjetivos (pos='a'), sustantivos (pos='n'), adverbios (pos='r') y verbos (pos='v')

Lemmatización

- EJERCICIO
  - Construir una función que a partir de un texto de entrada (ya separado en tokens) lematice su contenido de manera secuencial (primero considera que es nombre, a continuación considera que es verbo, adjetivo y/o adverbio)

Eliminación de palabras irrelevantes

- NLTK incorpora una librería para eliminar palabras irrelevantes (*stopwords*), esto es, que no contribuyen al significado del texto: “de”, “la”, “que”, “el”, “en”, “y”, “a”, “los”, “del”, …

```
from nltk.corpus import stopwords
stopwords.words('spanish')
```

- EJERCICIO: Crear una función que elimine las palabras irrelevantes de un texto tokenizado.

Corrección de texto

- Se puede utilizar el diccionario Wordlist para buscar aquellas palabras del texto que no están en el diccionario.

```
def unusual_words(text):
        text_vocab = set(w.lower() for w in text if w.isalpha())
        english_vocab = set(w.lower() for w in nltk.corpus.words.words())
        unusual = text_vocab.difference(english_vocab)
        return sorted(unusual)
```

Corrección de texto

- La función se puede probar de la siguiente manera:

```
# Número de palabras no usuales
print len(nltk.corpus.gutenberg.words('austen-sense.txt'))
print len(unusual_words(nltk.corpus.gutenberg.words('austen-sense.txt')))

# Número de palabras no usuales tras lematización
texto=nltk.corpus.gutenberg.words('austen-sense.txt')
wnl = nltk.WordNetLemmatizer()
clean_text=[wnl.lemmatize(t) for t in texto]
print len(unusual_words(clean_text))

clean_text=improved_lemmatizer(texto)
print unusual_words(clean_text)[:10]
print len(unusual_words(clean_text))
```

Corrección de texto

- EJERCICIO: Modificar la función anterior para que elimine las palabras inusuales de un texto tokenizado

Reemplazo de palabras por expresiones regulares

- Funcionalidad pensada para sustituir contracciones (en inglés) por expresiones regulares
- Podemos crear la clase RegexpReplacer y generar una librería replacers.py:

```
replacement_patterns = [
(r'won\'t', 'will not'),
(r'can\'t', 'cannot'),
(r'i\'m', 'i am'),
(r'ain\'t', 'is not'),
(r'(\w+)\'ll', '\g<1> will'),
(r'(\w+)n\'t', '\g<1> not'),
(r'(\w+)\'ve', '\g<1> have'),
(r'(\w+)\'s', '\g<1> is'),
(r'(\w+)\'re', '\g<1> are'),
(r'(\w+)\'d', '\g<1> would')
]

class RegexpReplacer(object):
    def __init__(self, patterns=replacement_patterns):
        self.patterns = [(re.compile(regex), repl) for (regex, repl) in patterns]

    def replace(self, text):
        s = text
        for (pattern, repl) in self.patterns:
            (s, count) = re.subn(pattern, repl, s)
        return s
```

Reemplazo de palabras por expresiones regulares

- Para usar esta función:

```
from replacers import RegexpReplacer
replacer = RegexpReplacer()

word_tokenize("can't is a contraction”)
['can’t', 'is', 'a', 'contraction']

word_tokenize(replacer.replace("can't is a contraction"))
['can', 'not', 'is', 'a', 'contraction']
```

Eliminación de caracteres repetidos

- Funcionalidad pensada para normalizar palabras que aparecen escritas con caracteres repetidos: ahhhhh, ummmmm,….
- Podemos crear la clase RepeatReplacer en la librería replacers.py:

```
class RepeatReplacer(object):
        def __init__(self):
                self.repeat_regexp = re.compile(r'(\w*)(\w)\2(\w*)')
                self.repl = r'\1\2\3’
        def replace(self, word):
                repl_word = self.repeat_regexp.sub(self.repl, word)
                if repl_word != word:
                        return self.replace(repl_word)
                else:
                        return repl_word
```

Eliminación de caracteres repetidos

- Para usar esta función:

```
from replacers import RepeatReplacer
replacer = RepeatReplacer()

print replacer.replace('looooove')
print replacer.replace('oooooh')
print replacer.replace('goose')
```

Eliminando sinónimos

- WordNet incorpora una base de datos léxica
- Para cada término proporciona una familia semántica de términos

```
from nltk.corpus import wordnet as wn
synset_clases=wn.synsets('pretty')

print synset_clases
[Synset('pretty.s.01'), Synset('pretty.s.02'),
Synset('reasonably.r.01')]
synset_clases[2].lemma_names
['reasonably', 'moderately', 'pretty', 'jolly', 'somewhat',
'fairly', 'middling', 'passably']
```

Eliminando sinónimos

- EJERCICIO: Construir una función que sustituya cada palabra de un texto tokenizado por el primer término de su primera familia semántica

Eliminando sinónimos

- Podemos probar esta función con:

```
text=nltk.corpus.gutenberg.words('austen-sense.txt')
print len(set(text))

clean_text1=improved_lemmatizer(text)
print len(set(clean_text1))
6833
clean_text2=[synonim_converter(w) for w in clean_text1]
print len(set(clean_text2))
5035
```

Un vocabulario avanzado

- Creemos una librería (lib_curso.py)

con todas las funciones:

  - improved_lemmatizer
  - remove_stopwords
  - elimina_unusual_words
  - synonim_converter?

- Creemos un script principal que haciendo uso de estas funciones, lea un texto, lo tokenize, lo limpie y normalice, y, finalmente, cree un bag of words
- Texto de ejemplo:
  - path = nltk.data.find('corpora/gutenberg/melville-moby_dick.txt')
  - raw = open(path, 'rU').read()

Clasificación de textos

Parte 4….

… próximamente en Matlab