GUÍA NLTK

Sistema de Almacenamiento y Recuperación de Información

NLTK (Natural Language Toolkit) es una librería de código abierto (bajo licencia Apache v2.0) para Python muy utilizada en los ámbitos de la recuperación de información y el procesamiento del lenguaje natural. Dispone de múltiples funciones para, entre otros, tokenización de documentos, clasificación de textos, análisis sintáctico y semántico. Además, cuenta con *wrappers* para varias librerías externas de procesamiento de lenguaje natural.

Corpus (colección de documentos)

Un corpus es una colección de documentos de texto. En muchas ocasiones los textos están anotados (tienen etiquetas asociadas a partes del texto) con información lingüística a diferentes niveles. Los corpora se suelen utilizan para aprender modelos computacionales utilizando algún algoritmo de aprendizaje automático.

Acceder a un corpus disponible en NLTK:

```
from nltk.corpus import nom_corpus
```

Obtener el texto del corpus:

```
nom_corpus.words()
```

Número de palabras que contiene el corpus:

```
len(nom_corpus.words())
```

Obtener oraciones del corpus:

```
nom_corpus.sents()
```

Número de oraciones que contiene un corpus:

```
len(nom_corpus.sents())
```

Obtener el nombre del primer fichero del corpus:

```
nom_corpus.fileids()[0]
```

Palabras que contiene el fichero n-ésimo:

```
text= nom_corpus.words(nom_corpus.fileids()[n-1])
```

Palabras que contiene un determinado fichero:

```
text= nom_corpus.words('nom_fichero.tbf')
```

Obtener las categorías de un corpus:

```
text= nom_corpus.categories()
```

Obtener las palabras de una categoría:

```
text= nom_corpus.words(categories=categoria)
```

Acceder a un corpus propio que no tenemos en NLTK (necesitamos 2 parámetros: ❶directorio donde se encuentra el corpus y ❷la lista de ficheros como [‘a.txt’, ‘test/b.txt’] o un patrón que empareje con todos los

ficheros, como ’[abc]/.*\.mrg’ que es una expresión regular. Más detalles sobre corpus en cap. 2 de http://nltk.org/book/):

```
from nltk.corpus import PlaintextCorpusReader
corpus_root = '/usr/share/dict' ❶
wordlists = PlaintextCorpusReader(corpus_root, '.*') ❷
print (wordlists.fileids())
print (wordlists.words('connectives'))
```

```
['README', 'connectives', 'propernames', 'web2', 'web2a', 'words']
['the', 'of', 'and', 'to', 'a', 'in', 'that', 'is', ...]
```

Más información en https://www.nltk.org/api/nltk.corpus.html#module-nltk.corpus

Estadísticas de un texto

NLTK dispone de una librería para calcular una serie de estadísticas sobre los textos.

Obtener las frecuencias de aparición de las palabras de un texto. Devuelve una lista de ítem, donde cada ítem es un par (key, value) donde key es la palabra y value es la frecuencia de aparición de la palabra.

```
from nltk.probability import *
fdist=FreqDist(texto)
```

Visualizar primeros 20 ítems (key, value), es decir (palabra, frecuencia):

```
print ("Primeros 20 ítems del fichero\n ", list(fdist.items())[:20])
```

Visualizar los pares (palabra, frecuencia) correspondientes a las 20 palabras más frecuentes:

```
print (fdist.most_common(20))
```

Obtener el vocabulario del texto anterior:

```
fdist.keys()
##o set(texto)
##o [w for w,f in fdist.most_common()]
```

Obtener la frecuencia de aparición de las palabras del fichero :

```
fdist.values()
```

Obtener la frecuencia de una palabra:

```
fdist['palabra']
## o fdist.freq('palabra')
```

Obtener el número de palabras que sólo aparecen una vez en el texto:

```
len([w for w in set(texto) if fdist[w]==1])
##o len(fdist.hapaxes())
```

Obtener la palabra más frecuente:

```
fdist.max()
```

Stopwords

Disponer de listas de palabras de parada o stopwords:

```
>>> from nltk.corpus import stopwords
>>> stopwords.words('spanish')
['de', 'la', 'que', 'el', 'en', 'y', 'a', 'los', 'del', 'se', 'las',
'por', 'un', …]
```

Función python que elimina stopwords:

```
>>> def remove_stopwords(text, language = 'english'):
...     # text es una lista de palabras
...     stopwords = nltk.corpus.stopwords.words(language)
...     result = [w for w in text if w.lower() not in stopwords]
...     return result
```

Función python que calcule que fracción de palabras en un texto no están en la lista de stopwords:

```
>>> def content_fraction(text, language = 'english'):
...     # text es una lista de palabras
...     stopwords = nltk.corpus.stopwords.words(language)
...     content = [w for w in text if w.lower() not in stopwords]
...     return len(content) / len(text)
...
>>> content_fraction(nltk.corpus.reuters.words())
0.65997695393285261
```

Stemmer/Lematizador (http://nltk.org/api/nltk.stem.html)

Módulo snowball :

Este módulo da acceso a los lematizadores Snowball desarrollados por Martin Porter. También hay una función de demostración que muestra los diferentes algoritmos. Se puede invocar directamente en la línea de comandos.

Desarrollado para 14 idiomas: Danish, Dutch, English, Finnish, French, German, Hungarian, Italian, Norwegian, Portuguese, Romanian, Russian, Spanish and Swedish.

Más información en http://snowball.tartarus.org/

class `nltk.stem.snowball.SnowballStemmer`(*language, ignore_stopwords=False*)

Se puede invocar siguiendo los pasos siguientes:

```
>>> from nltk.stem import SnowballStemmer
>>> print(" ".join(SnowballStemmer.languages)) # See which languages are
supported
danish dutch english finnish french german hungarian
italian norwegian porter portuguese romanian russian
spanish swedish
>>> stemmer = SnowballStemmer("german") # Choose a language
>>> stemmer.stem("Autobahnen") # Stem a word
'autobahn'
```

Crear una instancia del stemmer Snowball para una lengua específica.
Parámetros:

- **language** (*str or unicode*) – The language whose subclass is instantiated.
- **ignore_stopwords** (*bool*) – If set to True, stopwords are not stemmed and returned unchanged. Set to False by default.

Spanish Snowball stemmer:

class `nltk.stem.snowball.SpanishStemmer`(*language, ignore_stopwords=False*)

Bases: `nltk.stem.snowball._StandardStemmer`
variables:

- **__vowels** – Spanish vocales.
- **__step0_suffixes** – Sufijos a borrar en la etapa 0 del algoritmo.
- **__step1_suffixes** – Sufijos a borrar en la etapa 1
- **__step2a_suffixes** – Sufijos a borrar en la etapa 2a **__step2b_suffixes** – Sufijos a borrar en la etapa 2b
- **__step3_suffixes** – Sufijos a borrar en la etapa 3

Una descripción detallada del algoritmo se puede ver en
http://snowball.tartarus.org/algorithms/spanish/stemmer.html
Para obtener el stem de una palabra:
`stem`(*word*) donde word será str o Unicode y devuelve el stem de la palabra (tipo unicode)

English Snowball stemmer:

class `nltk.stem.snowball.EnglishStemmer`(*language, ignore_stopwords=False*)

Bases: `nltk.stem.snowball._StandardStemmer`

variables:

- **__vowels** – English vocales.
- **__double_consonants** – dobles consonantes.
- **__li_ending** – Letras que deben aparecer directamente antes del final de una palabra 'li'.
- **__step0_suffixes** – Sufijos a borrar en la etapa 0 del algoritmo.
- **__step1a_suffixes** – Sufijos a borrar en la etapa 1a.
- **__step1b_suffixes** – Sufijos a borrar en la etapa 1b.
- **__step2_suffixes** – Sufijos a borrar en la etapa 2.
- **__step3_suffixes** – Sufijos a borrar en la etapa 3.
- **__step4_suffixes** – Sufijos a borrar en la etapa 4.
- **__step5_suffixes** – Sufijos a borrar en la etapa 5.
- **__special_words** – Un diccionario conteniendo palabras las cuales tienen que ser especialmente tratadas para realizar el stemming.

Una descripción detallada del algoritmo se puede ver en
http://snowball.tartarus.org/algorithms/english/stemmer.html
Para obtener el stem de una palabra:
`stem`(*word*) donde word será str o Unicode y devuelve el stem de la palabra (tipo unicode)

Módulo `wordnet:`

Lematiza usando WordNet, devuelve la palabra sin cambiar si ésta no se encuentra en WordNet (sólo para english).

```
>>> from nltk.stem import WordNetLemmatizer
>>> wnl = WordNetLemmatizer()
>>> print(wnl.lemmatize('dogs'))
dog
>>> print(wnl.lemmatize('churches'))
church
>>> print(wnl.lemmatize('aardwolves'))
aardwolf
>>> print(wnl.lemmatize('abaci'))
abacus
>>> print(wnl.lemmatize('hardrock'))
hardrock
```

`lemmatize`(*word, pos='n'*)

Tokenizer

http://nltk.org/api/nltk.tokenize.html

WhitespaceTokenizer

```
s.split()
```

SpaceTokenizer

```
s.split(' ')
```

```
from nltk.tokenize import word_tokenize, wordpunct_tokenize, sent_tokenize
[word_tokenize(t) for t in sent_tokenize(s)]
```

Punkt Sentence Tokenizer

```
import nltk.data
sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
print('\n-----\n'.join(sent_detector.tokenize(text.strip())))
print('\n-----\n'.join(sent_detector.tokenize(text.strip(),
realign_boundaries=True)))
```

```
from nltk.tokenize.punkt import PunktWordTokenizer
PunktWordTokenizer().tokenize(s)
```

Module regexp

```
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer('\w+|\$[\d\.]+|\S+')
tokenizer = RegexpTokenizer('\s+', gaps=True)
capword_tokenizer = RegexpTokenizer('[A-Z]\w+')
tokenizer.tokenize(s)
```

Module treebank

```
from nltk.tokenize import TreebankWordTokenizer
```