PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE

FACULTAD DE LETRAS

Introducción a la lingüística computacional

César Antonio Aguilar

Facultad de Lenguas y Letras

28/09/2017

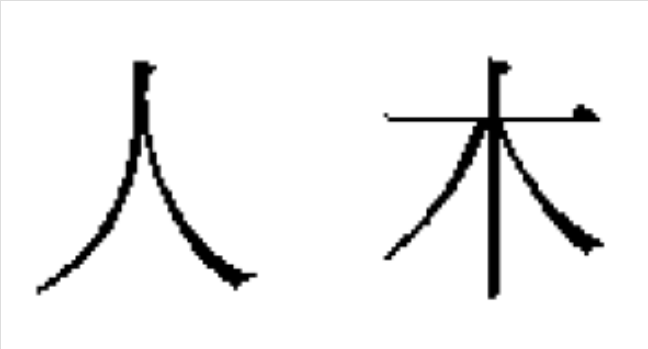
Cesar.Aguilar72@gmail.com

¿Qué es una *palabra* dentro de un corpus? (1)

Un hecho que se da muchas veces por sentado en lenguaje natural es el término *palabra.* Usualmente, la vemos como una cadena de caracteres delimitada por espacios en blanco, con un contenido léxico específico, además de tener rasgos fonológicos, morfológicos y sintácticos.

Ahora, ¿esto es así de claro siempre? ¿Qué pasa con las lenguas que tienen como medio de escritura los ideogramas?

El primer grafo se pronuncia *rén*, y significa “persona”; el segundo se prouncia *mù*, y significa “árbol”. Ambos son identificados como “palabras”.



¿Qué es una *palabra* dentro de un corpus? (2)

Otro caso, por ejemplo, son las lenguas con morfología aglutinante, p. e., en alemán una frase como *La biblioteca de la Universidad de Stuttgart equivalen a* dos palabras: *Universitätsbibliothek Stuttgart .*

Existen casos curiosos como la palabra *Bundeslandwirtschaftsministerin,* que equivale en español a *Ministro de Estado para Asuntos Económicos.*

¿Qué es una *palabra* dentro de un corpus? (3)

Incluso en español tenemos casos curiosos. ¿Cómo escribirían estas *palabras*?:

1. "Corre, ve y diles"/ *Correveidiles*.

2. ¿Así mismo/asimismo/a sí mismo?

3. "TVO luego, TKS".

4. Singular = "Hipótesis; plural = "Hipótesis".

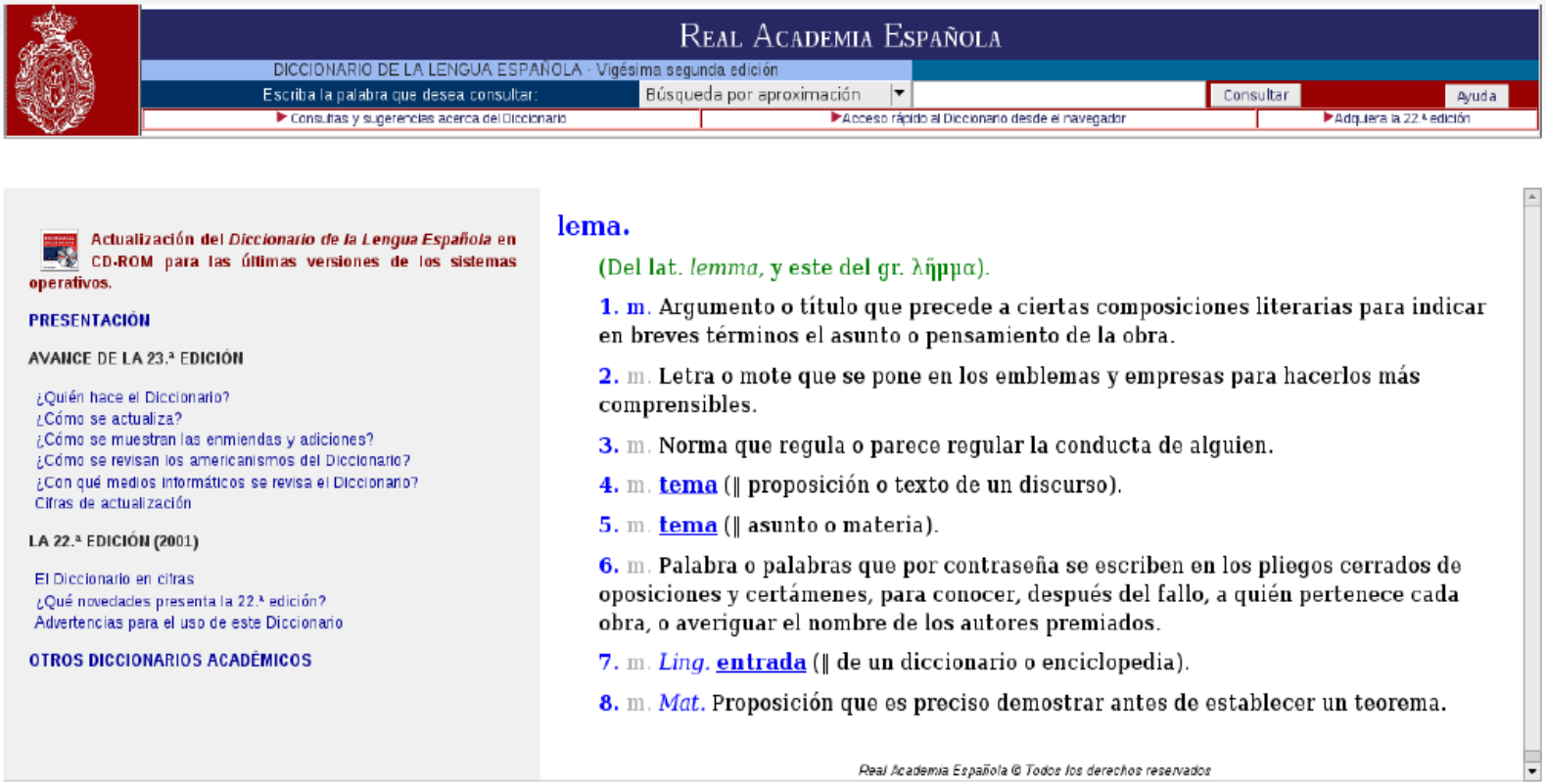
5. Alberto Ruy Sánchez/ Alberto Ruysánchez.

Y otros ejemplos más que quieran imaginar.

Análisis morfológico de palabras (1)

Dada esta complejidad, nos conviene saber algunas sobre morfología:

1. En morfología, *palabra* es un concepto vago, por lo que se usa el término **lexema** para describir el "esqueleto" de una palabra: *escribo*, *escribí*, *escribiré*, etc., son formas derivadas del lexema *escribir*. Cuando vemos un lexema en diccionario, lo denominamos **lema** (Ing. *lemma*), dando a entender que es una forma canónica.



Análisis morfológico de palabras (2)

2. Los lexemas sufren cambios, los cuales: i) o son necesarios debido a condiciones de concordancia en el contexto de una oración; ii) o son producto de modificaciones históricas. En concreto, hay dos cambios que nos son útiles en nuestras tareas:

Flexión: todas aquellas combinaciones que se hacen entre **raíces** (Ing. *stem/root*), y **afijos**, tales como **marcaciones léxicas** (género, número, tiempo, persona, etc.), **clíticos** (p. e.: *El maestro decidió explicar**les** esos conceptos*), o **partículas** (p. e.: *Peter**'s** book* = "El libro de Pedro").

Formación de palabras: todas aquellas otras combinaciones que se dan cuando se agrupan dos o más léxemas o raíces, que es el caso de la **composición** (p. e.: *sacacorchos*, *hombre-rana*, *quehacer*, etc.); o la **derivación**, que es la transformación que sufre un lexema cuando se relaciona con todo el paradigma de flexión de una lengua (p. e.: *fax* = *faxes*, *faxeado*, *faxear*, *faxeando*, *"por favor, faxéame este documento"*...).

Análisis morfológico de palabras (3)

En PLN, estos conceptos de morfología son aplicados en varias tareas. Veamos de momento dos básicas: tokenización y lematización.

- **Tokenización:** es el reconocimiento automático de palabras dado un tipo específico.
- **Lematización:** es la relación que tiene alguna forma flexionada o derivada con su raíz.

Forma	Lema	Etiqueta
chico	chico	NCMS000
chicos	chico	NCMP000
chica	chica	NCFS000
chicas	chica	NCFP000
oyente	oyente	NCCS000
oyentes	oyente	NCCP000
cortapapeles	cortapapeles	NCMN000
tesis	tesis	NCFN000
Antonio	antonio	NP00000

Identificando raíces y lemas (1)

Otra tarea que podemos realizar con nuestros documentos antes de etiquetarlos, es reconocer raíces de palabras, con miras a concretar una tarea que se conoce como **lematización,** es decir: identificar las formas canónicas de las palabras conforme a su registro en el diccionario**.** Esta tarea ha tenido un avance considerable a partir de la implementación del **algoritmo de Porter.**

Martin Porter desarrolló un algoritmo que, automáticamente, permite identificar asociar la raíz (o stemmer) de un conjunto de palabras similares. La descripción general de lo que hace este algortimo la pueden ver en la página WEB de Porter:

http://tartarus.org/~martin/PorterStemmer/def.txt

Identificando raíces y lemas (2)

Para ahorrarnos unas palabras, veamos la explicación que da de este algortimo Dan Jurafsky en su curso:

Porter's algorithm

- Commonest algorithm for stemming English
  - Results suggest it's at least as good as other stemming options
- Conventions + 5 phases of reductions
  - phases applied sequentially
  - each phase consists of a set of commands
  - sample convention: *Of the rules in a compound command, select the one that applies to the longest suffix.*

Identificando raíces y lemas (3)

Las reglas que aplica Porter son las siguientes:

Typical rules in Porter

- *sses* → *ss*
- *ies* → *i*
- *ational* → *ate*
- *tional* → *tion*

- Weight of word sensitive rules
- *(m>1) EMENT*
  - *replacement* *replac*
  - *cement* *cement*

Identificando raíces y lemas (4)

El algoritmo permite trabajar con palabras que tengan flexión regular o irregular:

Regulars and Irregulars

- Ok so it gets a little complicated by the fact that some words misbehave (refuse to follow the rules)
  - Mouse/mice, goose/geese, ox/oxen
  - Go/went, fly/flew
- The terms regular and irregular will be used to refer to words that follow the rules and those that don't.

Identificando raíces y lemas (5)

Finalmente, este algoritmo nos permite hacer un análisis morfológico bastante eficaz, incluso adaptándolo a otras lenguas.

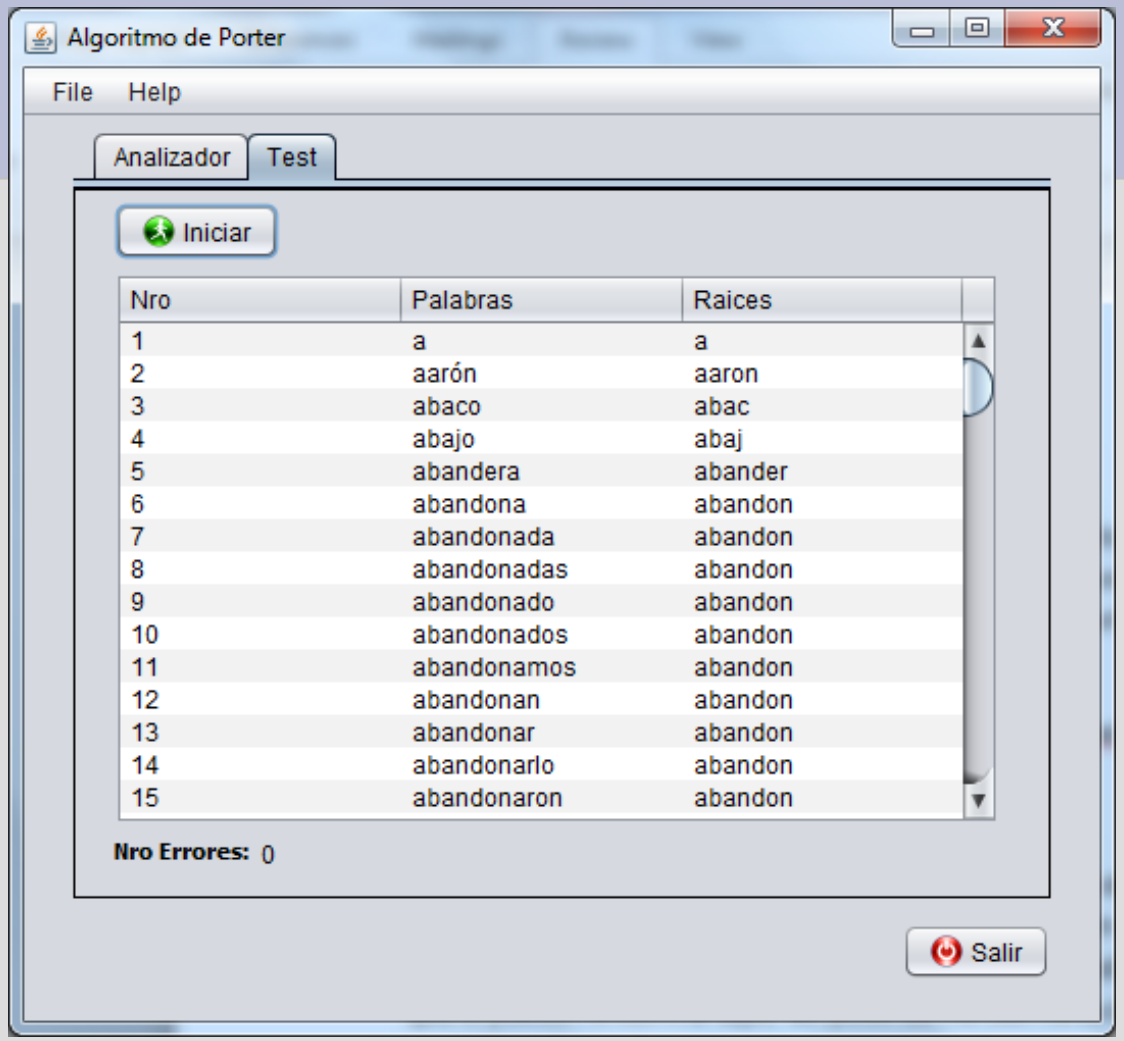
Morphological Parsing: Goal

English		Spanish		
Input	Morphologically Parsed Output	Input	Morphologically Parsed Output	Gloss
cats	cat +N +PL	pavos	pavo +N +Masc +Pl	'ducks'
cat	cat +N +SG	pavo	pavo +N +Masc +Sg	'duck'
cities	city +N +Pl	bebo	beber +V +PInd +1P +Sg	'I drink'
geese	goose +N +Pl	canto	cantar +V +PInd +1P +Sg	'I sing'
goose	goose +N +Sg	canto	canto +N +Masc +Sg	'song'
goose	goose +V	puse	poner +V +Perf +1P +Sg	'I was able'
gooses	goose +V +1P +Sg	vino	venir +V +Perf +3P +Sg	'he/she came'
merging	merge +V +PresPart	vino	vino +N +Masc +Sg	'wine'
caught	catch +V +PastPart	lugar	lugar +N +Masc +Sg	'place'
caught	catch +V +Past			

Figure 3.2 Output of a morphological parse for some English and Spanish words. Spanish output modified from the Xerox XRCE finite-state language tools.

Identificando raíces y lemas (6)

Para el español, existen versiones de este algoritmo, las cuales pueden revisar en el siguiente sitio:



http://snowball.tartarus.org/algorithms/spanish/stemmer.html

El algoritmo de Porter en NLTK (1)

NLTK nos permite lematizar usando dos algoritmos: el de Porter y el algoritmo **Lancaster** (una base de datos léxica). En el primer caso, la instrucción para importarlo es:

```
porter = nltk.PorterStemmer()
```

Para usar el otro, la instrucción es:

```
lancaster = nltk.LancasterStemmer()
```

El algoritmo de Porter en NLTK (2)

Para aplicar estos algoritmos, vamos a ocupar uno de nuestros textos, en este caso el artículo de Mira Ariel. Para ello, nuestras tradicionales líneas de código son:

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

Tokens_Ariel01 = nltk.word_tokenize(Cadena_Ariel01)
```

El algoritmo de Porter en NLTK (3)

Apliquemos entonces el algoritmo de Porter con la siguiente instrucción:

```
Lemma_Ariel01 = [porter.stem(t) for t in Tokens_Ariel01]

Rango_Lemmas01 = Lemmas_Ariel01[0:100]

print(Rango_Lemmas01)
```

Parte del resultado lo podemos ver en la siguiente lámina:

El algoritmo de Porter en NLTK (4)

IPython

File Edit View Kernel Magic Window Help

```
In [16]: print(Rango_Lemmas01)
['THE', 'DEVELOP', 'OF', 'PERSON', 'AGREEMENT', 'MARKER', ':', 'FROM', 'PRONOUN', 'TO',
'HIGHER', 'ACCESS', 'MARKERS*', 'Mira', 'Ariel', '(', '1999', ')', 'Tel-Aviv', 'Univers', '1',
'.', 'From', 'free', 'pronoun', 'to', 'verbal', 'agreement', ':', 'Introduct', 'Grammat',
'item', 'often', 'begin', 'their', 'linguist', 'life', 'as', 'regular', 'lexic', 'item', '(',
'Meillet', '1912', ')', '.', 'Grammatic', 'is', 'said', 'to', 'have', 'occur', 'when', 'the',
'posit', 'of', 'such', 'lexem', '(', 'or', 'even', 'phrase', ')', 'becom', 'fix', ',', 'their',
'mean', 'is', 'generalized/bleach', ',', 'their', 'domain', 'of', 'applic', 'enlarg', ',',
'significantli', 'rais', 'their', 'frequenc', ',', 'their', 'form', 'modifi', '(', 'usual',
'phonet', 'reduc', ')', ',', 'and', 'their', 'occurr', 'made', 'obligatori', ',', 'even',
'when', 'inform']
```

El algoritmo de Porter en NLTK (5)

Ahora, usemos el algoritmo de Lancaster:

```
Lemma_Ariel02 = [lancaster.stem(t) for t in
Tokens_Ariel01]

Rango_Lemmas02 = Lemmas_Ariel02[0:100]

print(Rango_Lemmas02)
```

El resultado lo podemos ver en la siguiente lámina:

El algoritmo de Porter en NLTK (6)

```
In [20]: print(Rango_Lemmas02)
['the', 'develop', 'of', 'person', 'agr', 'mark', ':', 'from', 'pronoun', 'to', 'high',
'access', 'markers*', 'mir', 'ariel', '(', '1999', ')', 'tel-aviv', 'univers', '1', '.', 'from',
'fre', 'pronoun', 'to', 'verb', 'agr', ':', 'introduc', 'gram', 'item', 'oft', 'begin', 'their',
'lingu', 'lif', 'as', 'regul', 'lex', 'item', '(', 'meillet', '1912', ')', '.', 'gram', 'is',
'said', 'to', 'hav', 'occur', 'when', 'the', 'posit', 'of', 'such', 'lexem', '(', 'or', 'ev',
'phrases', ')', 'becom', 'fix', ',', 'their', 'mean', 'is', 'generalized/bleach', ',', 'their',
'domain', 'of', 'appl', 'enlarg', ',', 'sign', 'rais', 'their', 'frequ', ',', 'their', 'form',
'mod', '(', 'us', 'phonet', 'reduc', ')', ',', 'and', 'their', 'occur', 'mad', 'oblig', ',',
'ev', 'when', 'inform']
```

¿Qué diferencias encuentran respecto a los resultados arrojados por el algoritmo de Porter?

El algoritmo de Porter en NLTK (7)

Finalmente, se puede tokenizar un texto empleando como diccionario la red léxica **WordNet**. A través de este sitio pueden revisar qué es WordNet.

http://wordnet.princeton.edu/perl/webwn

La instrucción es:

```
wnl_Ariel01 = nltk.WordNetLemmatizer()

Lemma_Ariel03 = [wnl_Ariel01.lemmatize(t) for t in Tokens_Ariel01]

Rango_Lemmas02 = Lemmas_Ariel02[0:100]

print(Rango_Lemmas02)
```

El algoritmo de Porter en NLTK (7)

Y nuestro resultado es:

IPython

File Edit View Kernel Magic Window Help

```
In [25]: print(Rango_Lemmas03)
['THE', 'DEVELOPMENT', 'OF', 'PERSON', 'AGREEMENT', 'MARKERS', ':', 'FROM', 'PRONOUNS', 'TO',
'HIGHER', 'ACCESSIBILITY', 'MARKERS*', 'Mira', 'Ariel', '(', '1999', ')', 'Tel-Aviv',
'University', '1', '.', 'From', 'free', 'pronoun', 'to', 'verbal', 'agreement', ':',
'Introduction', 'Grammatical', 'item', 'often', 'begin', 'their', 'linguistic', 'life', 'a',
'regular', 'lexical', 'item', '(', 'Meillet', '1912', ')', '.', 'Grammaticization', 'is',
'said', 'to', 'have', 'occurred', 'when', 'the', 'position', 'of', 'such', 'lexeme', '(', 'or',
'even', 'phrase', ')', 'becomes', 'fixed', ',', 'their', 'meaning', 'is',
'generalized/bleached', ',', 'their', 'domain', 'of', 'applicability', 'enlarged', ',',
'significantly', 'raising', 'their', 'frequency', ',', 'their', 'form', 'modified', '(',
'usually', 'phonetically', 'reduced', ')', ',', 'and', 'their', 'occurrence', 'made',
'obligatory', ',', 'even', 'when', 'informationally']
```

Gracias por su atención

Blog del curso:
http://cesaraguilar.weebly.com/introduccioacuten-a-la-linguumliacutestica-computacional.html