

# **Modelos de Inteligencia Artificial**

*Conforme a contenidos del «Curso de Especialización  
en Inteligencia Artificial y Big Data»*



**Modelos de  
Inteligencia\_Artificial**

**Universidad de Castilla-La Mancha**

Escuela Superior de Informática  
Ciudad Real



# Índice general

---

<b>1. Caracterización de sistemas de Inteligencia Artificial</b>	<b>1</b>
1.1. Fundamentos de la IA . . . . .	1
1.1.1. Características principales . . . . .	1
1.1.2. Tipos de inteligencia . . . . .	4
1.1.3. Historia de la IA . . . . .	5
1.2. Campos de aplicaciones . . . . .	7
1.2.1. Agricultura: cultivo óptimo . . . . .	7
1.2.2. Banca: detección de fraude . . . . .	8
1.2.3. Ciberseguridad: detección y tratamiento de ataques . . . . .	8
1.2.4. Atención sanitaria: diagnóstico de pacientes . . . . .	9
1.2.5. Logística: rutas y optimización . . . . .	9
1.2.6. Telecomunicaciones: optimización de redes . . . . .	10
1.2.7. Juegos: creación de agentes inteligentes . . . . .	10
1.2.8. Arte: creatividad . . . . .	11
1.3. Técnicas de la Inteligencia Artificial . . . . .	11
1.3.1. Algoritmos de búsqueda . . . . .	12
1.3.2. Algoritmos evolutivos . . . . .	14
1.3.3. Aprendizaje automático ( <i>machine learning</i> ) . . . . .	15
1.4. Nuevas formas de interacción . . . . .	18
<b>2. Utilización de los modelos de la Inteligencia Artificial</b>	<b>21</b>
2.1. Entornos de trabajo . . . . .	22

2.2.	Sistemas de resolución de problemas . . . . .	23
2.2.1.	Tipos de problemas en entornos de trabajo . . . . .	23
2.2.2.	Programas agente . . . . .	24
2.3.	Modelos de sistemas de IA . . . . .	32
2.3.1.	Planificación automática . . . . .	32
2.3.2.	Sistemas de razonamiento impreciso . . . . .	39
<b>3.</b>	<b>Procesamiento de Lenguaje Natural</b>	<b>47</b>
3.1.	Contexto del Procesamiento de Lenguaje Natural . . . . .	47
3.2.	Preprocesamiento de textos . . . . .	48
3.2.1.	Identificación del idioma . . . . .	49
3.2.2.	Eliminación de secciones/elementos no relevantes . . . . .	50
3.2.3.	Limpieza y normalización de términos . . . . .	50
3.2.4.	Corrección de errores . . . . .	53
3.2.5.	Palabras vacías . . . . .	53
3.2.6.	Lematización . . . . .	54
3.2.7.	Stemming . . . . .	55
3.2.8.	Segmentación . . . . .	56
3.3.	Análisis léxico . . . . .	58
3.4.	Análisis sintáctico . . . . .	59
3.5.	Análisis semántico . . . . .	60
<b>4.</b>	<b>Aplicaciones del Procesamiento de Lenguaje Natural</b>	<b>65</b>
4.1.	Introducción . . . . .	65
4.2.	Aplicaciones del Lenguaje Natural a la Recuperación de Información	65
4.2.1.	Datos vs. Información . . . . .	66
4.2.2.	Modelos de Recuperación de Información . . . . .	68
4.2.3.	Arquitectura de un Buscador . . . . .	73
4.2.4.	Medidas de evaluación . . . . .	75
4.3.	Aplicaciones del Procesamiento Lenguaje Natural a la Clasifica- ción de Textos . . . . .	76
4.3.1.	Support Vector Machines . . . . .	77
<b>5.</b>	<b>Análisis de Sistemas Robotizados</b>	<b>83</b>
5.1.	Simulador de Robótica CoppeliaSim . . . . .	85
5.2.	Vistazo general a la interfaz del simulador . . . . .	85

---

5.3. Propiedades de los objetos . . . . .	87
5.4. Geometría Pura vs Geometría no pura . . . . .	89
5.5. Jerarquización de objetos . . . . .	89
5.6. Articulaciones . . . . .	90
5.7. Diseño de un robot móvil básico . . . . .	91
5.8. Diseño de un brazo robótico básico . . . . .	98



# Listado de acrónimos

---

# 3

## Capítulo

# Procesamiento de Lenguaje Natural

---

Jesús Serrano Guerrero

## 3.1. Contexto del Procesamiento de Lenguaje Natural

El Procesamiento de Lenguaje Natural es una disciplina que nace en los años 60 íntimamente relacionada con el mundo de la Inteligencia Artificial y la Computación Lingüística, pero también relacionada con otras áreas como la Recuperación de Información, la Computación Afectiva, la Psicología, etc. Delimitar los límites donde comienza una y terminan las otras es demasiado complejo dado que unas dependen de las otras.

El proceso del análisis del lenguaje se puede descomponer en distintas fases distinguiendo, de forma más pedagógica, entre tres componentes principales: sintáctica, semántica y pragmática. Grosso modo, primero habría que analizar qué frases o sentencias se han escrito o dicho, qué quieren decir, y qué implicaciones pragmáticas conllevan. Sin embargo, analizando más en detalle sería necesario añadir primero unas fases de análisis previo sobre qué elementos conforman cada una de esas sentencias y qué funcionalidades desarrollan. La Figura 3.1 muestra interconectadas todas las fases necesarias para llegar a comprender todas las implicaciones de un texto.



**Figura 3.1:** Fases del Procesamiento de Lenguaje Natural.



También, es necesario remarcar que el análisis del texto es una tarea fundamental, y de ella se derivan otras tareas relacionadas como, por ejemplo, la tarea de generar texto automáticamente (Natural Language Generation, en inglés). Esta tarea es más compleja que la anterior, haciendo uso de ella, pero ofrece un gran número de funcionalidades en la actualidad como se está viendo en bots automáticos como Alexa de Google o Siri de Apple. Además, para que uno de estos bots puede responder a una pregunta, primero debe entender qué es lo que ha querido decir el usuario, es decir, cuál es el fin de sus preguntas o intervenciones, por lo tanto, es necesario recurrir a esa componente pragmática que la que se hablaba anteriormente. Esta tarea se conoce como Comprensión del Lenguaje Natural (Natural Language Understanding en inglés), es decir, el conocimiento de la intención del usuario, qué es lo que quiere, necesita o pretende.

A continuación, se comentarán cada una de las fases fundamentales de Procesamiento de Lenguaje Natural.

## 3.2. Preprocesamiento de textos

Antes de comenzar a analizar cualquier texto, la primera tarea que debe hacerse es definir claramente las unidades lingüísticas bien definidas que conformarán dicho texto. En este sentido, se puede hablar de (I) caracteres, que serían las unidades más pequeñas, (II) palabras o términos, que estarían compuestas por términos, y (III) frases o sentencias, que serían el resultado de varias palabras conjuntas, o incluso (IV) párrafos entendidos como agrupaciones de frases. La fase de preprocesamiento se encarga de analizar el texto, principalmente contenido en un archivo binario o de texto plano representado por distintos símbolos pertenecientes al algún estándar de codificación de caracteres como ASCII, ASCII-Extendido o Unicode, entre otros. Así pues, lo primero a realizar es la decodificación de dicho texto. Una vez se han identificado los símbolos, es importante detectar el idioma al que pertenecen estos símbolos, ello permitirá en las siguientes fases, extraer la información contenida con mayor precisión.



El sistema de codificación es muy importante dados los múltiples símbolos que pueden representarse: árabes, chinos, japoneses, europeos, símbolos típicos de Internet, etc.

Los documentos no necesariamente están compuestos exclusivamente por texto, sino otros elementos binarios como imágenes pueden ser contenidos. Por ello, es necesario diseccionar el documento, eliminando aquellos elementos que no guardan relación estricta con el texto. Finalmente, el texto es segmentado en sus unidades básicas, términos o frases.

La segmentación se denomina técnicamente tokenización (tokenization en inglés). El objetivo de este proceso es determinar dónde empieza y dónde termina un término (o frase). Técnicamente, la unidad básica con la que se trabaja se denomina token, es decir, a cada término representa un token. Una vez, los tokens son deli-

mitados, es necesario buscar la forma canónica normalizada, dado que un mismo token puede ser escrito de distintas maneras, pero se refiere a un único concepto. Así, por ejemplo, los tokens Dr., dr., doctor, Doctor se refieren a la persona que ejerce la medicina o tiene estudios de doctorado, sin embargo, sus graffas son distintas, por lo que es necesario buscar una forma única que los represente a todos. Por ejemplo, podría elegirse simplemente el término “doctor” para representarlos a todos.



El análisis de tokens puede ser sensible desde el punto de vista del uso de mayúsculas y minúsculas, así los tokens *Doctor* y *doctor* son diferentes, incluso no necesariamente tiene porque tener la misma forma canónica cuando tienen significados diferentes.

Como puede verse, son muchas tareas que hay que realizar cuando se preprocesa un texto. Estas serían las principales tareas:

- Identificación de la codificación de los caracteres.
- Identificación del idioma del texto.
- Limpieza y normalización.
- Eliminación de secciones no textuales.
- Corrección de errores.
- Lematización.
- Stemming.
- Segmentación del texto.
  - Tokenización o segmentación de términos.
  - Segmentación de frases.

### 3.2.1. Identificación del idioma

Es importante conocer en qué idioma está escrito un texto. Eso permitirá la identificación de ciertos elementos que darán información para entender lo que hay escrito. Por ejemplo, dependiendo del idioma se utilizarán distintos logogramas, es decir, las unidades mínimas escritas que pueden representar un concepto. Un ejemplo sería el símbolo € que representaría el concepto *euro*. No obstante, en otros idiomas como el chino cada símbolo puede tener asociado un concepto. Por otro lado, hay idiomas que pueden ser silábicos, donde cada símbolo representa una sílaba, o alfabéticos, como el español, donde cada símbolo suele estar asociado a un único sonido cuando se pronuncia. Es por ello importante, saber el sistema de codificación de caracteres que se están usando, no todos permiten representar todos los caracteres.

Existen idiomas que tienen delimitadores para sílabas, palabras o frases. Algunos los tienen todos, pero algunos solo tienen uno de ellos, lo cual, a veces, hace muy difícil identificar lo que es una palabra, sílaba o frase.

Con el fin de ver un ejemplo práctico, Python provee una librería para Procesamiento de Lenguaje Natural muy completa llamada NLTK (Natural Language ToolKit) con la que se implementarán gran parte de los ejemplos sobre este tema. Su instalación e importación es sencilla como se puede ver en el Listado 3.1.

#### Listado 3.1: Instalación y descarga del paquete básico de funcionalidades de NLTK

```
1 !pip install nltk
2 import nltk
3 nltk.download("popular")
```

Sin embargo, NLTK no dispone de funciones para detectar idiomas. Por ello, es necesario recurrir a otras funcionalidades disponibles que podrían ser útiles como el paquete *languagedetect* que se usa en el Listado 3.2 para detectar el idioma de varias frases.

#### Listado 3.2: Ejemplo de detección de idiomas en varios lenguajes

```
1 !pip install langdetect
2 from langdetect import detect
3 detect("Spain is different")
4 detect("España es diferente")
```

### 3.2.2. Eliminación de secciones/elementos no relevantes

Es necesario entender el contexto donde se encuentre el texto que se desea analizar. De esta manera, parte de la contenido del que se dispone puede ser irrelevante y es necesario eliminarlo. Documentos como artículos periodísticos, libros, etc. están organizados por secciones, capítulos, titulares.... que pueden ser fácilmente identificables y pueden ser relevantes o no dependiendo de la aplicación que se quiera implementar. Uno de los casos más llamativos serían las páginas web. En ellas, el texto deseado se encuentra integrado entre miles de etiquetas en lenguajes como HTML que deben ser eliminadas primeramente, como se puede ver en la Figura 3.2.

### 3.2.3. Limpieza y normalización de términos

Como ya se ha comentado anteriormente, es necesario determinar lo que es un token, pero también es necesario eliminar todos aquellos elementos que no son relevantes como acentos, guiones, signos de puntuación, etc. y normalizarlos, es decir, crear una normal común para representarlos, por ejemplo, eliminando la diferencia entre palabras escritas en mayúsculas o minúsculas, etc.

```
<p>Según las ideas publicadas acerca de esta estación, lo que se busca es <strong>una estación
<!-- BREAK 3 -->
<p>Se trataría de <strong>una estructura "ultragrande que abarca kilómetros"</strong> y sería '
<!-- BREAK 4 --> <div class="ad ad-mid">
  <div class="ad-box" id="div-gpt-m-lat">
    <script>
      googletag.cmd.push(function() { googletag.display("div-gpt-m-lat"); });
    </script>
  </div>
</div>
```


Figura 3.2: Ejemplo de texto contenido entre etiquetas de marcado web.

Para la detección de muchas de estas estructuras que es necesario limpiar o normalizar, se suelen utilizar expresiones regulares que permiten detectar palabras con características especiales. Por ejemplo, si en un tweet se quiere eliminar un mención hecha mediante el símbolo '@' (@trump), pero mantener la palabra a la que se refiere (trump), sería necesario delimitar la expresión que caracterice a todas las posibles menciones. En forma de regla gramatical, una posible solución sería una expresión similar a esta: @[a-zA-Z0-9]+, es decir, primero debe aparecer obligatoriamente el símbolo '@' y luego vendrá un número indeterminado de mayúsculas, minúsculas o números que conformarán el *nickname* de la persona que se menciona.

Son muchas las tareas que pueden ser necesarias para normalizar y limpiar las palabras de un texto. Es por ello que, en muchas ocasiones, será necesario manejar las funcionalidades a nivel de gestión de cadenas de texto. Por ejemplo, algunas de las funcionalidades principales que pueden ser interesantes son:

split(t)	Devuelve todos los términos borrando la expresión t
replace (t, u)	Reemplaza la expresión t por u dentro de un texto
lower()	Escribe en minúsculas un término
upper()	Escribe en mayúsculas un término
islower()	Devuelve si una palabra está escrita en minúsculas
isupper()	Devuelve si una palabra está escrita en mayúsculas
startswith(t)	Indica si una palabra comienza por la expresión t
endswith(t)	Indica si una palabra termina con la expresión t
isalpha()	Indica si un término está formado solo por caracteres alfabéticos
isalnum()	Indica si un término está formado por términos alfanuméricos
isdigit()	Indica si un término es un dígito

Tabla 3.1: Algunas funciones básicas del tipo str en Python



**Ejercicio:** Implementar un programa que detecte si una palabra es un enlace o no. Ejemplos de palabras: marea, <https://www.marca.es>, [www.as.com](http://www.as.com), terremoto, [es.kiosko.net](http://es.kiosko.net), [httpcasa.com](http://httpcasa.com), [www.nosoy.es](http://www.nosoy.es)



**Ejercicio:** Dado el siguiente tweet, generar su vocabulario, es decir, los términos que lo conforman. Para ello, será necesario eliminar todos los signos de puntuación menos aquellos que tiene una función numérica, poner todas las palabras en minúsculas.

**Tweet:** El #terremoto que sacudió el sábado al suroeste de #Haití dejó 2.189 #muertos, 12.000 heridos y 32 desaparecidos, según cifras oficiales actualizadas al alza la noche del miércoles, mientras se mantienen las operaciones de búsqueda y rescate.

Como se ha mencionado, en caso de existir algún tipo especial de palabra o expresión que desea ser buscada, se puede recurrir al uso de patrones. Para ello, existe la librería *re* (*regular expressions*) que permite la detección de patrones en Python. Primeramente, es necesario definir el patrón que seguirán los términos buscados, y luego se buscará si los términos corresponden o no con ese patrón. Un caso sencillo puede verse en el Listado 3.3 que permite descubrir qué extensiones pertenecen a archivos de imágenes o no, sobre un grupo previo definido como primer argumento de la función *match* en la línea ⑤.


Listado 3.3: Ejemplo de búsqueda de patrones usando la librería *re*

```
1 import re
2 extensiones = ['jpg', 'png', 'gif', 'mp3', 'doc']
3
4 for tipoarchivo in extensiones:
5     if re.match('jpg|png|gif|bmp', tipoarchivo):
6         print('La extensión ', tipoarchivo, 'es un formato de imagen')
7     else:
8         print('La extensión ', tipoarchivo, 'no es un formato de imagen')
```

También se puede detectar qué páginas pertenecen a un dominio concreto, como se ve en el Listado 3.4 mediante la función *search*.

Listado 3.4: Ejemplo de búsqueda de patrones usando la librería *re*

```
1 lista_url = ['http://www.aaa.es',
2             'ftp://www.aaa.es',
3             'http://www.bbb.es']
4 for elemento in lista_url:
5     if re.search('es$', elemento):
6         print(elemento)
```

Más información sobre esta funcionalidad puede ser encontrada en  Link: <http://docs.python.org/3/library/re.html>.

### 3.2.4. Corrección de errores

Muchas veces, al escribir deprisa se cometen errores. El mejor ejemplo sería Internet, y especialmente las redes sociales, donde pueden encontrarse miles de palabras abreviadas o mal escritas, que hace muy difícil su detección e interpretación. Un ejemplo de ello puede verse en el tweet de la figura 3.3, que contiene faltas de ortografía, elongaciones de términos (*likeoo*), etc.

Una herramienta típica del preprocesamiento es la corrección de palabras mal deletreadas. La función encargada normalmente recibe el término técnico de *spell-checker*.



**Figura 3.3:** Ejemplo de tweet cuyo contenido es complejo de procesar.

La mayoría de algoritmos que se encargan de este tipo de errores usan aproximaciones semánticas, es decir, intentan buscar términos similares en diccionarios u otros recursos que guarden cierta similitud con la palabra deletreada. Un ejemplo de spell-checker puede verse en el Listado 3.5, el cual trata de corregir una frase mal escrita mediante la librería *TextBlob*.

**Listado 3.5:** Ejemplo de corrección de términos mal deletreados mediante la librería *TextBlob*

```
1 !pip install textblob
2 from textblob import TextBlob
3
4 str = "whaat ixs yoor nami"
5
6 new_doc = TextBlob(str)
7 result = new_doc.correct()
8 print(result)
```

### 3.2.5. Palabras vacías

El concepto de palabra vacía (*stopword* en inglés) se refiere a aquellos términos que no se consideran relevantes para la aplicación desarrollada, y por consiguiente, se pueden eliminar. Normalmente, las palabras que aportan poco información suelen ser artículos, preposiciones, verbos auxiliares (ser, estar...) o copulativos, fórmulas modales (can, could, may,...), etc.

Si en la frase “la casa estaba vacía” se elimina el verbo copulativo *estar* y la preposición *la*, el resultado “casa vacía”, semánticamente es similar, pero el vocabulario se ha reducido a la mitad.

Existen distintos tipos de algoritmos para eliminar las palabras vacías, pero los más habituales simplemente se basan en aproximaciones estáticas, es decir, dada una lista predeterminada de palabras del idioma correspondiente, se eliminan todas ellas. También, existen aproximaciones dinámicas, que analizan el contexto de los textos a trabajar, y detectan aquellos términos que son irrelevantes, para posteriormente eliminarlos.

Normalmente, la mayoría de librerías traen una serie de palabras vacías diseñadas a priori. En NLTK hay varias colecciones predeterminadas según el idioma, tal y como se puede ver en el Listado 3.6.

#### Listado 3.6: Stopwords típicas en español en NLTK

```
1 from nltk.corpus import stopwords
2 stopwords.words("spanish")
```

### 3.2.6. Lematización

Para entender el concepto de lematización es necesario recurrir del concepto de sufijo. Los sustantivos *guerras*, *guerrilla*, *guerrero*, *guerrera*, *guerrillero*, *guerrillera*, *guerreros*, *guerreras*, ..., evidentemente, tienen algo en común, su raíz, todas provienen de la misma idea: *guerra*. Las diferentes variantes de la palabra *guerra* se forman mediante la adición de sufijos que pueden dar lugar a formas plurales u otros sustantivos de la misma familia.

Con los verbos, ocurre algo similar. La conjugación del verbo **amar** da lugar a múltiples términos: **amaré**, **amáramos**, **amaríamos**, **amáramos**, **amé**...

El proceso de lematización consiste en descubrir cuál es el lema, o palabra origen de la que provienen el resto de palabras. En este último ejemplo, podría decirse que el lema puede ser *amar* o el nombre *amor*. Es necesario elegir una palabra como representante de esa familia, que es lo que se denomina lema.

La lematización permite la reducción del vocabulario con el que se trabaja. Si en un texto se encuentran los términos *amaríamos* y *amáramos*, ambos se pueden sustituir por un único término *amor*.

Los algoritmos de lematización también pueden tener un carácter sintáctico y semántico, con el fin de entender cuál es el significado de una palabra. Por ejemplo, dada la palabra *amo* es posible indicar un sustantivo, personas que posee algo, o el hecho de amar, en cuyo caso sería un verbo. Para poder entender estas funcionalidades léxicas y los significados asociados, normalmente se requiere de algún diccionario o tesauro especializado como Wordnet, del que se hablará más adelante. Un ejemplo usando dicho recurso puede verse en el Listado 3.7.

Listado 3.7: Ejemplo de lematización de la palabra *dogs* usando Wordnet

```
1 from nltk.stem import WordNetLemmatizer
2
3 lemmatizer = WordNetLemmatizer()
4
5 print(lemmatizer.lemmatize("dogs"))
```

Como es evidente, el diccionario debe estar en el idioma que se necesita, por lo que en caso de no disponer de uno, normalmente se evita su uso o se opta por la aplicación de las técnicas de *stemming*.

### 3.2.7. Stemming

El proceso de stemming es similar al de lematización pero en este caso, es un poco más radical. En lugar de obtenerse la palabra origen o lema, se obtiene cuál es la raíz común (o stem en inglés) de un conjunto de términos. En los ejemplos anteriores, las raíces podrían ser “guerr” para las palabras relacionados con “guerra” o “am” para las palabras relacionadas con “amar”. Los algoritmos de stemming suelen ser mucho más sencillos de implementar que los de lematización, pero pueden presentar mayores inconvenientes. El hecho de recortar demasiado una raíz se llama *overstemming*, muchas otras palabras puede encajar con la raíz “am”, por lo que no podría ser únicamente representativa de “amar”. También, puede ocurrir el efecto contrario, *understemming*, es decir, que el algoritmo no recorte lo suficiente la raíz y las palabras no se identifiquen unívocamente con esa raíz.



Obviamente los procesos de stemming y lematización suponen una simplificación de la información pero, también conlleva asociada una pérdida de información. El desarrollador debe valorar si es conveniente o no su uso, según el tipo de aplicación que desarrolle.

En la librería NLTK pueden encontrarse varios algoritmos de stemming. En el Listado 3.8, dos de los algoritmos más famosos pueden verse: *Porter* y *Lancaster*. La función *stem* (líneas ⑥ y 10) permite extraer la raíz de cada término.

Listado 3.8: Ejemplo de algoritmos típicos de stemming

```
1 texto = "We are living special moments"
2 tokens = nltk.word_tokenize(texto)
3
4 # Este es posiblemente el algoritmo más famoso
5 porter = nltk.PorterStemmer()
6 print ([porter.stem(t) for t in tokens])
7
8 # Este es otro de los algoritmos más famosos
9 lancaster = nltk.LancasterStemmer()
10 print ([lancaster.stem(t) for t in tokens])
```





**Ejercicio.** Probar los algoritmos anteriores sobre diferentes textos con el objetivo de ver cuál de los dos algoritmos selecciona raíces más pequeñas y reflexionar sobre las limitaciones y ventajas de cada aproximación.

### 3.2.8. Segmentación

La segmentación o tokenización consiste en la búsqueda de las unidades básicas de un texto o tokens. Estas dependen del tipo de idioma. Idiomas como el chino o japonés tienen una serie de características que lo hacen más complejo de procesar dado que no es fácil conocer dónde están los límites de una frase o una palabra. Por ello, es necesario conocer en detalle la estructura léxica y morfológica de las palabras y frases. Este apartado se centrará en aquellos idiomas donde la delimitación se puede establecer a través de espacios en blanco como el español o el inglés.

La segmentación normalmente suele ser a dos niveles, a nivel de palabra o de sentencia.

#### Nivel de palabra

Para detectar las unidades mínimas con las que trabajar o tokens, es necesario trabajar en coordinación con un proceso de limpieza y de normalización de tokens.

El proceso de limpieza conllevaría la eliminación de caracteres que no aportan información pero pueden aparecer acompañando a una palabra. Por ejemplo, el uso de comillas simples o dobles ('amigo', "amigo") debe tratarse si se quiere obtener la palabra final *amigo*. Otro ejemplo serían los símbolos como las interrogaciones o puntos que pueden aparecer junto a una palabra. En la frase *¿cuál es tu nombre?*, si solo se atiende a la separación por espacios, quedarían como términos *¿cuál* y *nombre?* En este caso, habría que borrar los caracteres de interrogación.

El proceso de normalización también conlleva la eliminación de mayúsculas si es el caso. En la frase "La casa no es la esperada.", el artículo "la" aparece dos veces, con distinta grafía, por lo que es necesario normalizarla, normalmente utilizando el término en minúsculas. Hay que tener cuidado porque no siempre es necesario sustituir los términos en mayúsculas por términos en minúsculas, pueden tener un significado especial, como en el caso de los nombres propios (Ej: *Pedro*).

La mayoría de las palabras están separadas por un espacio en blanco, pero pueden encontrarse algunos problemas como el uso de apostrofes o guiones, con los que hay que tomar decisiones. Por ejemplo, la palabra *Fórmula-1* puede entenderse como un único token, o como dos, *fórmula* y *uno*. En inglés, *la casa de Pedro* se traduciría como *Pedro's home*. En este último caso sería necesario detectar las estructuras léxicas y decidir qué se interpreta como un token y qué no.

Existen muchos más casos como las abreviaturas como Sr. que podrían ser sustituida por el token *señor* o U.S.A. por *United States of America*. O todos aquellos términos relacionados con fechas o cifras numéricas, que podrían ser considerados como multi-términos. La fecha 1/1/1900 podría ser normalizada por los tokens *uno de enero de mil novecientos*, o la cifra 3,5€ podría ser normalizada con la expresión *tres euros y cincuenta céntimos* o *tres euros y medio*. No hay una solución específica, será el desarrollador el que tome la decisión según las necesidades de su aplicación.

Para ver un ejemplo de tokenización por palabras, el Listado 3.9 muestra la función `word_tokenize` que se encarga de buscar todas las palabras separadas por espacios dentro de un fragmento de texto.

**Listado 3.9: Ejemplo de tokenización por palabras con NLTK**

```
1 from nltk.tokenize import word_tokenize
2 EJEMPLO_PALABRAS = "Hola mi nombre es Pepe"
3 word_tokenize(EJEMPLO_PALABRAS)
```

## Nivel de frase

El otro nivel importante de segmentación consiste en la detección de la frase como unidad de información. Para poder llevar a cabo posteriormente el análisis sintáctico y semántico, es necesario reconocer lo que es una frase. Para ello, se suelen detectar separados como el punto (seguido o final), pero dada la complejidad del lenguaje, otros separadores pueden aparecer como las comas, punto y coma, símbolos de interrogación o exclamación, puntos suspensivos, entrecomillado, entre otros.



Conocer el sistema de codificación de caracteres permite, por ejemplo, conocer cuáles son los delimitadores de una frase, como pueden ser los símbolos de puntuación en español (.,;¿?!) o cuándo empiezan las mayúsculas. Por ejemplo, sistemas como UNICODE agrupan todos los símbolos de presentación de mayúsculas seguidas, por lo que es fácil detectarlas. En idiomas como el árabe no hay diferencias entre mayúsculas y minúsculas.

La complejidad del idioma hace que muchas veces sea difícil delimitar lo que es una frase o no. Existen sentencias sencillas como “La case está vacía.”. Sin embargo, otras frases como, por ejemplo, la oración subordinada “No ha llegado, así que me voy.” podría dividirse en otras más sencillas como: “No ha llegado”, “me voy”.

Un ejemplo sencillo de tokenizador de frases puede ser el presentado en el Listado 3.10.

Listado 3.10: Ejemplo de tokenización por frases con NLTK

```
1 from nltk.tokenize import sent_tokenize
2 EJEMPLO_FRASES = "Hola mi nombre es Pepe. Mi casa no estoy seguro de dónde está."
3 sent_tokenize(EJEMPLO_FRASES)
```

### 3.3. Análisis léxico

Una vez se han extraído las unidades básicas con las que se puede trabajar, ahora es necesario analizarlas para poder entender un poco más sus funcionalidades a nivel léxico. Para ver qué función desempeña un término en una frase, existe una tarea que se llama *etiquetación de partes del discurso* o POS (Part of Speech) tagging. Los términos pueden desarrollar funciones como: nombre común (*casa*), nombre propio (*Pedro*), verbo (*correr*), adverbio (*alegremente*), preposiciones (*hasta*), etc. Para anotar dichas funcionalidades de forma estandarizada, existen una serie de etiquetas universales que implementan la mayoría de herramientas y que son las que se listan a continuación:

- ADJ: adjetivo
- ADP: adposición
- ADV: adverbio
- AUX: auxiliar
- CCONJ: conjunción coordinada
- DET: determinante
- INTJ: interjección
- NOUN: nombre
- NUM: número
- PART: partícula
- PRON: pronombre
- PROPN: nombre propio
- PUNCT: puntuación
- SCONJ: conjunción subordinada
- SYM: símbolo
- VERB: verbo
- X: otro

Conocer cuál es la función de una palabra dentro de la frases, puede ser útil dependiendo de la aplicación. Así, si se quiere conocer el estado de ánimo de persona, será lógico analizar los adjetivos (ADJ) usados en sus frases: triste, contento, entusiasmado. Sin embargo, si se quiere conocer el valor de un producto, será más lógico analizar los símbolos (SYM: euro, dolares.....) y las cantidades numéricas (NUM) asociadas. Estas etiquetas son universales, pero pueden subdividirse en otras más descriptivas que permiten diferenciar entre cada una de las clases anteriores. Por ejemplo, un nombre puede estar en singular (NN) o plural (NNS), o un verbo puede estar sin conjugar (VB) o en gerundio (VBG). A modo el ejemplo, en el siguiente Listado 3.11, se identifican todas las etiquetas asociadas a cada término de una frase.

Listado 3.11: Detección de etiquetas del discurso

```
1 text = word_tokenize("And from now on this will be completely different")
2 nltk.pos_tag(text)
```



**Ejercicio.** Identificar en un texto todas las cantidades numéricas encontradas borrando el resto de términos.

### 3.4. Análisis sintáctico

Según Noam Chomsky existen distintos tipos de gramáticas para describir los lenguajes formales:

- Gramáticas de tipo 0 o sin restricciones
- Gramáticas de tipo 1 o sensible al contexto
- Gramáticas de tipo 2 o libres del contexto
- Gramáticas de tipo 3 o regulares.

Dependiendo del tipo de gramática con la que se haya generado el lenguaje con el que se trabaje, el proceso del análisis sintáctico será más o menos complicado. Analizar lenguajes como HTML donde cada una de las expresiones sigue unas reglas muy concretas por usar gramáticas sensibles al contexto, es relativamente sencillo dado que las etiquetas tienen una forma y parámetros bien definidos como se ve en la Tabla 3.2. Sin embargo, el lenguaje natural está regido por gramáticas sensibles al contexto, que son más complejas y dan lugar a mayores ambigüedades.

Para analizar sintácticamente un lenguaje, es necesario implementar una herramienta que recibe el nombre de *parser*. Su objetivo es analizar cada una de las sentencias que conforman el texto, y para ello, normalmente se recurre a su representación gráfica en forma de *árbol sintáctico*.

```
<html>
<head>
<title>Escribir el título </title>
</head>
<body>
Aquí aparecerá el contenido de la web
</body>
```

Tabla 3.2: Ejemplo de código HTML bien estructurado

Este árbol tiene forma invertida, la raíz aparece en la parte superior y las hojas en la parte inferior. Cada una de las ramas del árbol pudiera representar distintos tipos de oraciones (coordinadas, subordinadas, disyuntivas, etc.), mientras las hojas representan las funciones léxicas de cada términos, o lo que se llamó anteriormente, etiquetas POS (Parts of Speech). Un ejemplo de árbol puede verse en la figura 3.4 para la frase “The refugees in Afghanistan escaped mainly towards Madrid”.

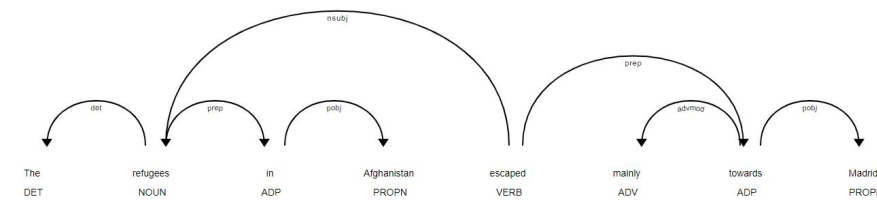


Figura 3.4: Ejemplo de árbol sintáctico.

Una vez que se han extraído todas las etiquetas POS de una frase, puede verse si una frase o una parte de una frase se corresponde con un patrón determinado o no, es decir, es un tipo de frase concreta o no que puedan ser relevante para la aplicación desarrollada. Este proceso se denomina *chunking*. Por ejemplo, se puede intentar buscar subfrases o fragmentos con el patrón o chunk *DT NN IN JJ*, es decir, un determinante, un nombre, una preposición y un adjetivo. Así, la estructura nominal *el hombre de negro* o *la mujer de rojo* serían detectados por dicho patrón.

### 3.5. Análisis semántico

Una vez analizadas las frases desde el punto de vista sintáctico, es necesario entender qué es lo que quieren decir. Para ello, se recurre al nivel semántico, que permitirá entender el significado tanto de los términos individuales como todos ellos funcionando dentro de una misma oración.

**WordNet Search - 3.1**  
 - [WordNet home page](#) - [Glossary](#) - [Help](#)

Word to search for:

Display Options:

Key: "S:" = Show Synset (semantic) relations, "W:" = Show Word (lexical) relations  
 Display options for sense: (gloss) "an example sentence"

**Noun**

- [S:](#) (n) **home**, [place](#) (where you live at a particular time) "*deliver the package to my home*"; "*he doesn't have a home to go to*"; "*your place or mine?*"
- [S:](#) (n) [dwelling](#), **home**, [domicile](#), [abode](#), [habitation](#), [dwelling house](#) (housing that someone is living in) "*he built a modest dwelling near the pond*"; "*they raise money to provide homes for the homeless*"
- [S:](#) (n) **home** (the country or state or city where you live) "*Canadian tariffs enabled United States lumber companies to raise prices at home*"; "*his home is New Jersey*"

**Figura 3.5:** Resultado de la búsqueda de la palabra *home* en Wordnet.

Para poder entender el significado de un término, normalmente, se recurre al uso de recursos externos bien elaborados. Entre este tipo de recursos caben destacar los diccionarios, los lexicones, los tesauros, y las ontologías.

El concepto de diccionario es ampliamente conocido, un conjunto de términos junto con su definición asociada. Un tesoro puede ser entendido como una lista estructurada de términos o conceptos cuyo objetivo es representar de forma unívoca el contenido de los documentos y de las consultas dentro de un conjunto documental. Es especialmente útil para los procesos de indización de los documentos, es decir, el almacenamiento estructurado de los archivos de texto, así como los procesos búsqueda de información en dicha colección documental. Este tipo de recursos, no solo contiene el conjunto de términos característicos de la colección de textos con la que se trabaje, sino que además, puede contener definiciones de los mismos, frases en las que puedan aparecer para demostrar los distintos contextos en los que pueden aparecer los términos, así como términos relacionados como:

- Sinónimos: Términos con el significado pero distinta grafía: **recibo - factura**.
- Hiperónimos: Términos que engloban a otros. Por ejemplo, *alimento* es hiperónimo de *carne* y *pescado*.
- Hipónimos: Consecuentemente, *carne* y *pescado* son hipónimos de *alimento*.

Posiblemente el más famoso sea Wordnet<sup>1</sup>, cuya interfaz con distintas definiciones para el término *home* puede verse en la figura 3.5.

<sup>1</sup> Link: <http://wordnetweb.princeton.edu/perl/webwn>

El concepto de ontología está más relacionado con el mundo de la Informática, y en concreto con la Inteligencia Artificial. Se puede definir como la descripción explícita de una conceptualización. Dado un dominio de trabajo, la ontología representa el esqueleto del conocimiento subyacente. Su representación suele realizarse en forma de grafo de términos entre los que se establecen múltiples relaciones, estableciendo la semántica del dominio tratado. Dado que es un artefacto informático, suele estar representado mediante algún lenguaje estándar como el lenguaje de la Web Semántica denominado OWL (Ontology Language (OWL)).

El uso de recursos de este tipo puede permitir a las aplicaciones realizar tareas semánticas como desambiguar una palabra. Por ejemplo, dada una palabra como “régimen”, analizando las palabras que aparecen a su alrededor, es decir, su contexto, puede saberse si se está hablando de un “régimen político”, “un régimen alimenticio”, “un régimen de visitas”, etc. En el Listado 3.12 puede verse un ejemplo de uso de Wordnet, en el que se pueden sacar todos los conjuntos de significados que tiene la palabra *big*, y ver uno a uno, aunque en el ejemplo solo se ve el primero de ellos, cuál es la definición que da el tesoro (ver línea ③) y además, unos ejemplos del contexto donde se podría encontrar la palabra *big* según ese significado (ver línea ④).

Listado 3.12: Detección de etiquetas del discurso

```
1 from nltk.corpus import wordnet
2 synsets=wordnet.synsets('big')
3 print(synsets[0].definition())
4 print(synsets[0].examples())
```



**Ejercicio.** Buscar en WordNet todos los significados para el término *doctor* e identificar a qué significado se refiere dicho término en la frase: "The doctor talked to his patient".

Es necesario remarcar que aunque se use la idea de *término* por simplicidad, al estar en el nivel semántico, podría hablarse de concepto, y también, debería hablarse del concepto de *collocations* o conjuntos de palabras que tienen sentido cuando van conjuntamente. Por ejemplo, *darse una ducha*, *día festivo* o *corredor de bolsa* son conceptos que deberían ser preprocesados con el fin de ser detectados, ya que si se interpretan como términos individuales, sin contexto, pierden su significado.

Finalmente, una vez que se han visto los niveles léxico, sintáctico y semántico, sería necesario mencionar que también existiría un nivel superior, que sería el nivel *pragmático*. Una vez analizadas todas las frases de un texto, sería necesario entender cuál es la intencionalidad de las mismas, para ello, muchas veces es necesario recurrir al contexto. Una frase como “Me he quedado con tu reloj”, indica la el cambio de posesión de un objeto; sin embargo, la frase “Me he quedado con tu cara” no indica la posesión de un objeto sino la intención de intimidar

una persona. Llegar a comprender las frases con este nivel de detalle, requiere de técnicas muy avanzadas relacionadas con otras disciplinas como la Computación Afectiva (Affective Computing en inglés) o el Razonamiento de Sentido Común (Common-sense Reasoning en inglés).