Procesamiento de Lenguaje Natural

Pre-procesamiento – Análisis Léxico

Qué vamos a ver hoy?

Pre-procesamiento – Análisis Léxico

- Pre-procesamiento, para qué?
- Normalización.
- Tokenización.
- Lemmatización.
- Stemmer.
- Qué incluir en el pipeline?

Pre-procesamiento

- El pre-procesamiento consiste en una serie de pasos para limpiar y estandarizar el texto en una forma que pueda ser consumida por otros sistemas.

- La idea principal es eliminar el contenido no necesario en uno o más textos para obtener contenido más limpio y fácilmente procesable.

Pre-procesamiento

- El pre-procesamiento consiste en una serie de pasos para limpiar y estandarizar el texto en una forma que pueda ser consumida por otros sistemas.

- La idea principal es eliminar el contenido no necesario en uno o más textos para obtener contenido más limpio y fácilmente procesable.

Segmenting Tokenizing	Case conversion	Eliminar caracteres especiales
Corregir spelling	Remover términos innecesarios	Expansión de contracciones

Pre-procesamiento

Eliminar HTML Tags

- El texto no estructurado contiene una gran cantidad de ruido, especialmente si fue obtenido *scrapeando* de la web.

- Los tags HTML, JavaScript o Iframe típicamente no agregan valor para el análisis y entendimiento del texto.

```
<p><b>Londres</b> (en <a href="/wiki/Idioma_ingl%C3%A9s" title="Idioma inglés">inglés</a>:
<i>London</i>, <span class="nounderlines" style="white-space:nowrap;"><span class="unicode"><a
href="/wiki/Archivo:En-uk-London.ogg" title="Acerca de este sonido"><img alt="Acerca de este sonido"
src="//upload.wikimedia.org/wikipedia/commons/thumb/8/8a/Loudspeaker.svg/11px-Loudspeaker.svg.png"
decoding="async" width="11" height="11"
srcset="//upload.wikimedia.org/wikipedia/commons/thumb/8/8a/Loudspeaker.svg/17px-Loudspeaker.svg.png
1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/8/8a/Loudspeaker.svg/22px-Loudspeaker.svg.png 2x"
data-file-width="20" data-file-height="20" /></a> <a
href="//upload.wikimedia.org/wikipedia/commons/c/c3/En-uk-London.ogg" class="internal" title="En-uk-
London.ogg"><span title="Representación en el Alfabeto Fonético Internacional (IPA o AFI)"
class="IPA">ˈlʌndən</span></a>&#32;<small class="metadata audiolinkinfo" style="cursor:help;">(<a
href="/wiki/Ayuda:Multimedia" title="Ayuda:Multimedia"><span style="cursor:help;" title="¿Problemas al
reproducir este archivo?">?</span></a>·<a href="/wiki/Archivo:En-uk-London.ogg" title="Archivo:En-uk-
London.ogg"><span style="cursor:help;" title="Acerca de este
sonido">i</span></a>)</small></span></span>) es la capital y mayor ciudad de <a href="/wiki/Inglaterra"
title="Inglaterra">Inglaterra</a> y del <a href="/wiki/Reino_Unido" title="Reino Unido">Reino Unido</a>.
```

Londres es la capital y mayor ciudad de Inglaterra y del Reino Unido.

https://es.wikipedia.org/wiki/Londres

Pre-procesamiento

Conversión de case

- Modificar el case de las palabras u oraciones para facilitar algunas tareas, como encontrar coincidencias entre diferentes palabras, mientras a la vez se reduce el tamaño del vocabulario.

Upper

Lower

Title

Pre-procesamiento

Conversión de case

- Modificar el case de las palabras u oraciones para facilitar algunas tareas, como encontrar coincidencias entre diferentes palabras, mientras a la vez se reduce el tamaño del vocabulario.

Upper	Lower	Title
TODOS LOS CARACTERES DEL TEXTO ESTÁN EN MAYÚSCULA	todos los caracteres del texto están en minúscula	El Primer Carácter De Cada Palabra Está Capitalizado

Pre-procesamiento

Eliminar caracteres especiales

- Los símbolos y los caracteres especiales usualmente agregan ruido en texto no estructurado.
  - Caracteres no alfanuméricos.
  - Dígitos.

- Por lo general, se pueden usar expresiones regulares simples para eliminarlos.

Pre-procesamiento

Eliminar caracteres especiales

- Los símbolos y los caracteres especiales usualmente agregan ruido en texto no estructurado.
  - Caracteres no alfanuméricos.
  - Dígitos.

- Por lo general, se pueden usar expresiones regulares simples para eliminarlos.

```
pattern = r'[^a-zA-z0-9\s]'
```

Pre-procesamiento

Eliminar caracteres especiales

- Los símbolos y los caracteres especiales usualmente agregan ruido en texto no estructurado.
  - Caracteres no alfanuméricos.
  - Dígitos.

- Por lo general, se pueden usar expresiones regulares simples para eliminarlos.

- **Qué eliminar DEPENDE DEL PROBLEMA !!**
  - Por ejemplo, los símbolos de puntuación podrían ser útiles en las tareas de sentiment analysis u opinión.

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Una **expresión regular** (o *regex*) es un String que define un **patrón de búsqueda**.

- Cuatro tareas principales:
  - **Encontrar** una cadena en un texto más largo.
  - **Validar** que un String se adecua a un formato específico.
  - **Reemplazar** o **insertar** texto.
  - **Dividir** Strings.

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Consideremos un ejemplo simple donde queremos encontrar la palabra “`cool`” en un texto.
- Simplemente definir la expresión “cool” es suficiente.

```
NLP•is•cool
```

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Consideremos un ejemplo simple donde queremos encontrar la palabra “`cool`” en un texto.
- Simplemente definir la expresión “cool” es suficiente.

```
NLP•is•cool
```

- Pero, hay que destacar que no opera a nivel palabra, sino a nivel char.

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Consideremos un ejemplo simple donde queremos encontrar la palabra “`cool`” en un texto.
- Simplemente definir la expresión “cool” es suficiente.

`NLP • is • cool`

- Pero, hay que destacar que no opera a nivel palabra, sino a nivel char.

Expectativa	Realidad
`cool`	**`c o o l`**
Palabra `cool`	Primero `c`, luego una `o`, otra `o` y una `l`

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Consideremos un ejemplo simple donde queremos encontrar la palabra “`cool`” en un texto.
- Simplemente definir la expresión “cool” es suficiente.

```
NLP is cool
```

- Pero, hay que destacar que no opera a nivel palabra, sino a nivel char.
- Entonces, la expresión también matchearía con:

```
NLP is the coolest
She bought a watercooler
NLP is supercool
```

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Qué podemos matchear? Bloques básicos

- **Chars particulares.**
  - Podemos simplemente especificar un char cualquiera y matcheará con todas las apariciones en el texto.

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Qué podemos matchear? Bloques básicos

- **Chars particulares.**
  - Podemos simplemente especificar un char cualquiera y matcheará con todas las apariciones en el texto.

  - Por ejemplo, “a” va a matchear con todas las apariciones de a en el texto.

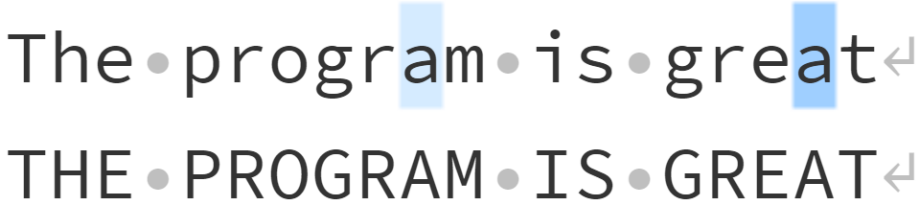
Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Qué podemos matchear? Bloques básicos

- **Chars particulares.**
  - Podemos simplemente especificar un char cualquiera y matcheará con todas las apariciones en el texto.

  - Por ejemplo, “a” va a matchear con todas las apariciones de a en el texto.



Es case-sensitive!

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Qué podemos matchear? Bloques básicos

- **Chars particulares.**
  - Podemos simplemente especificar un char cualquiera y matcheará con todas las apariciones en el texto.

- **Números.**
  - Dígitos del 0 al 9.

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Qué podemos matchear? Bloques básicos

- **Chars particulares.**
  - Podemos simplemente especificar un char cualquiera y matcheará con todas las apariciones en el texto.

- **Números.**
  - Dígitos del 0 al 9.
  - Por ejemplo, la expresión “3” va a matchear con cualquier 3 que encuentre.

Uso Python 3.7

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Qué podemos matchear? Bloques básicos

- **Chars particulares.**
  - Podemos simplemente especificar un char cualquiera y matcheará con todas las apariciones en el texto.

- **Números.**
  - Dígitos del 0 al 9.

- **Espacios en blanco.**
  - `\s`: espacio
  - `\t`: tab
  - `\n`: salto de línea
  - `\r`: carriage return

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Qué podemos matchear? Bloques básicos

- **Chars particulares.**
  - Podemos simplemente especificar un char cualquiera y matcheará con todas las apariciones en el texto.

- **Números.**
  - Dígitos del 0 al 9.

- **Espacios en blanco.**
  - `\s`: espacio
  - `\t`: tab
  - `\n`: salto de línea
  - `\r`: carriage return

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Además de poder encontrar secuencias definidas ad-hoc, existen algunas expresiones que permiten encontrar determinados conjuntos de caracteres.

`\w`	Hace matching con letras, dígitos y _.	`\s`	Hace matching con caracteres de espacios, incluidos los tabs.
`\W`	Hace matching con lo contrario de \w.	`\S`	Hace matching con lo contrario de \s.
`\d`	Hace matching con los dígitos.	`\n`	Hace matching con saltos de línea.
`\D`	Hace matching con lo contrario de \d.	`\r`	Hace matching con el carriage return.
`\t`	Hace matching con tabs.		

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Además de poder encontrar secuencias definidas ad-hoc, existen algunas expresiones que permiten encontrar determinados conjuntos de caracteres.

`\w`	Hace matching con letras, dígitos y _.	`\s`	Hace matching con caracteres de espacios, incluidos los tabs.
`\W`	Hace matching con lo contrario de \w.	`\S`	Hace matching con lo contrario de \s.
`\d`	Hace matching con los dígitos.		
`\D`	Hace matching con lo contrario de \d.		

Por ejemplo, `\d` matchearía:

En el caso de 412 son 3 matches y no solo 1.

```
Tengo 2 perros y 1 gato.
El precio del libro es 412.10
```

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Además de poder encontrar secuencias definidas ad-hoc, existen algunas expresiones que permiten encontrar determinados conjuntos de caracteres.

`\w`	Hace matching con letras, dígitos y _.	`\s`	Hace matching con caracteres de espacios, incluidos los tabs.
`\W`	Hace matching con lo contrario de \w.	`\S`	Hace matching con lo contrario de \s.
`\d`	Hace matching con los dígitos.		
`\D`	Hace matching con lo contrario de \d.		

Por ejemplo, `\D` matchearía:

```
Tengo 2 perros y un gato
El precio del libro es de 412.10
```

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Además de poder encontrar secuencias definidas ad-hoc, existen algunas expresiones que permiten encontrar determinados conjuntos de caracteres.

`\w`	Hace matching con letras, dígitos y _.	`\s`	Hace matching con caracteres de espacios, incluidos los tabs.
`\W`	Hace matching con lo contrario de \w.	`\S`	Hace matching con lo contrario de \s.
`\d`	Hace matching con los dígitos.		
`\D`	Hace matching con lo contrario de \d.		

Por ejemplo, `\w` matchearía:

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

- Además de poder encontrar secuencias definidas ad-hoc, existen algunas expresiones que permiten encontrar determinados conjuntos de caracteres.

`\w`	Hace matching con letras, dígitos y _.	`\s`	Hace matching con caracteres de espacios, incluidos los tabs.
`\W`	Hace matching con lo contrario de \w.	`\S`	Hace matching con lo contrario de \s.
`\d`	Hace matching con los dígitos.		
`\D`	Hace matching con lo contrario de \d.		

Por ejemplo, `\w` matchearía:

Por ejemplo, `\W` matchearía:

Pre-procesamiento

Expresiones regulares → Chars especiales

- Existe un conjunto de caracteres especiales que no pueden ser utilizados de forma directa en las regex.
  - Tienen significados especiales.
  - Deben ser “escapeados” precedidos por una “\”.
    - (al igual que los conjuntos que acabamos de ver)

```
. \ + * ? [ ^ ] $ ( ) { } ! < > | -
```

Pre-procesamiento

Expresiones regulares → Chars especiales

- Existe un conjunto de caracteres especiales que no pueden ser utilizados de forma directa en las regex.
  - Tienen significados especiales.
  - Deben ser “escapeados” precedidos por una “\”.
    - (al igual que los conjuntos que acabamos de ver)

. \ + * ? [^] $ ( ) { } ! < > | -

.	Matchea cualquier char except salto de línea.	( )	Agrupar chars o regex.
\	Permite matchear los conjuntos que vimos y otras expresiones de encoding.	!	Permite negar regex.
[ ]	Define clases de caracteres para el matching o un char específico. Dentro de [] casi todos los chars especiales son interpretados como no especiales.	<>	Establecen límites.
^	Matchea con el inicio de la cadena, o usado luego de [, permite negar la clase.	|	or.
$	Matchea con el fin de la cadena.	-	Denota un rango de chars.

Pre-procesamiento

Expresiones regulares → Chars especiales

- **Conjuntos de chars. [ ]**

Por ejemplo, `[aeiou]` matchea con cualquier vocal.

Por ejemplo, `[012356789]` matchea con cualquier dígito.

Pre-procesamiento

Expresiones regulares → Chars especiales

- **Conjuntos de chars. [ ]**

Por ejemplo, `[aeiou]` matchea con cualquier vocal.

Por ejemplo, `[012356789]` matchea con cualquier dígito.

No hace falta especificarlo por extension, sino que se pueden definir rangos.

`[012356789]` es lo mismo que `[0-9]`

Pre-procesamiento

Expresiones regulares → Chars especiales

- **Conjuntos de chars. [ ]**

Por ejemplo, `[aeiou]` matchea con cualquier vocal.

Por ejemplo, `[012356789]` matchea con cualquier dígito.

Por ejemplo, `[2-4]` matchea con 2, 3 o 4.

Pre-procesamiento

Expresiones regulares → Chars especiales

- **Conjuntos de chars. [ ]**

Por ejemplo, `[aeiou]` matchea con cualquier vocal.

También se puede hacer con las letras:

- [a-z] matchea con cualquier minúscula.
- [A-Z] matchea con cualquier mayúsucula.

Se pueden combinar chars especiales:

- [\s\d] matchea con un espacio en blanco o con un dígito

Pre-procesamiento

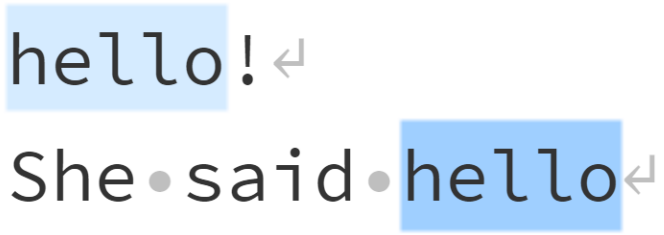
Expresiones regulares → Chars especiales

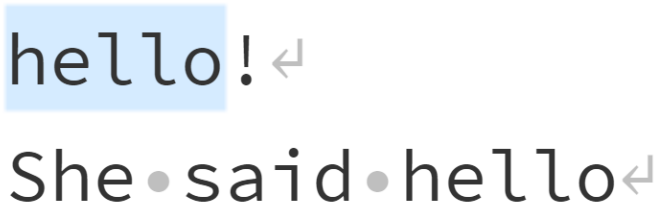
- **Anchors**
  - Al comienzo de la cadena: ^
  - Al finalizar la cadena: $

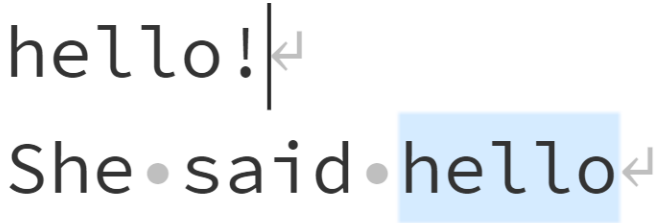
Pre-procesamiento

Expresiones regulares → Chars especiales

- **Anchors**
  - Al comienzo de la cadena: ^
  - Al finalizar la cadena: $



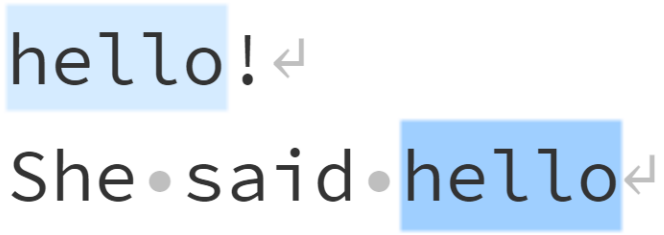




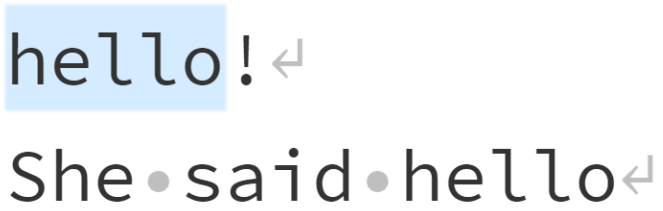
Pre-procesamiento

Expresiones regulares → Chars especiales

- **Anchors**
  - Al comienzo de la cadena: ^
  - Al finalizar la cadena: $



hello



^hello

hello$

Pre-procesamiento

Expresiones regulares → Chars especiales

- **Escapando chars**
  - Se utiliza la \ antes de cada char que se quiera interpreter de forma literal.

```
Mr.•Stark↵
```

```
Mr. Stark
```

Pre-procesamiento

Expresiones regulares → Chars especiales

- **Escapando chars**
  - Se utiliza la \ antes de cada char que se quiera interpreter de forma literal.

```
Mr. Stark
```

```
Mr.•Stark↵
MrX•Stark↵
Mr3•Stark↵
```

No era lo que se esperaba!

Pre-procesamiento

Expresiones regulares → Chars especiales

- **Escapando chars**
  - Se utiliza la \ antes de cada char que se quiera interpreter de forma literal.

`Mr. Stark`

```
Mr.·Stark↵
MrX·Stark↵
Mr3·Stark↵
```

No era lo que se esperaba!

`Mr\. Stark`

```
Mr.·Stark↵
MrX·Stark↵
Mr3·Stark↵
```

Pre-procesamiento

Expresiones regulares → Repetición de bloques

- Existe un conjunto de caracteres especiales que no pueden ser utilizados de forma directa en las regex.
  - Tienen significados especiales.
  - Deben ser “escapeados” precedidos por una “\”.
    - (al igual que los conjuntos que acabamos de ver)

```
. \ + * ? [ ^ ] $ ( ) { } ! < > | -
```

Cuantificadores

*	Cero o más	{n}	n ocurrencias
+	Uno o más	{n,}	n o más ocurrencias
?	Cero o uno	{n,m}	Entre n y m ocurrencias

Pre-procesamiento

Expresiones regulares → Repetición de bloques

- **Naïve.**
  - Repetimos el patrón tantas veces como querramos encontrar las coincidencias.

Pre-procesamiento

Expresiones regulares → Repetición de bloques

- **Naïve.**
  - Repetimos el patrón tantas veces como querramos encontrar las coincidencias.

- **Cuantificadores.**

  - **Repetición fija.**
    - Se usa {…} para indicar la cantidad de veces que un patron debe repetirse.

Pre-procesamiento

Expresiones regulares → Repetición de bloques

- **Naïve.**
  - Repetimos el patrón tantas veces como querramos encontrar las coincidencias.

- **Cuantificadores.**

  - **Repetición fija.**
    - Se usa `{…}` para indicar la cantidad de veces que un patron debe repetirse.
      - `\d\d` puede escribirse como `\d{2}`

Pre-procesamiento

Expresiones regulares → Repetición de bloques

- **Naïve.**
  - Repetimos el patrón tantas veces como querramos encontrar las coincidencias.

- **Cuantificadores.**

  - **Repetición fija.**
    - Se usa `{…}` para indicar la cantidad de veces que un patron debe repetirse.
      - `\d\d` puede escribirse como `\d{2}`

    - Se puede especifiar un rango de repeticiones.
      - `\d{2,4} matchea con números de 2 o 4 cifras.`

Pre-procesamiento

Expresiones regulares → Repetición de bloques

- **Naïve.**
  - Repetimos el patrón tantas veces como querramos encontrar las coincidencias.

- **Cuantificadores.**

  - **Repetición fija.**
    - Se usa `{…}` para indicar la cantidad de veces que un patron debe repetirse.
      - `\d\d` puede escribirse como `\d{2}`

    - Se puede especifiar un rango de repeticiones.
      - `\d{2,4} matchea con números de 2 o 4 cifras.`

Pre-procesamiento

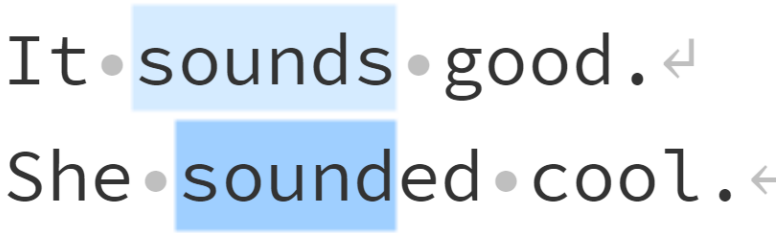
Expresiones regulares → Repetición de bloques

- **Naïve.**
  - Repetimos el patrón tantas veces como querramos encontrar las coincidencias.

- **Cuantificadores.**

  - **Repetición fija.**
    - Se usa `{…}` para indicar la cantidad de veces que un patron debe repetirse.

  - **Repetición flexible.**
    - `?`: indica 0 o 1 vez
      - Por ejemplo: `sounds?`
    - `+`: una o más veces
      - Por ejemplo: `\d+`
    - `*`: 0 o más veces



El ? solo afecta a la s.

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

match()

Intenta hacer matching entre la regex y el String desde el comienzo del String. Retorna un objeto match si lo encuentra, None si falla.

```
re.match('\w+', 'hello world!')
<re.Match object; span=(0, 5),
match='hello’>
```

findall()

Retorna una lista de todas las instancias de la regex en el String, ordenados de izquierda a derecha.

```
re.findall('[A-Z]\w+', 'hello World!')
['World’]
```

search()

Similar a match(), pero busca en todo el String.

```
re.search('or', hello World')
<re.Match object; span=(7, 9), match='or'>
```

split()

Divide el String de acuerdo a las ocurrencias de la regex.

```
re.split('\s+', 'hello world from duia')
['hello', 'world', 'from', 'duia']
```

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

`[\w]`	Matchear con todo menos con espacios.
`[\W]`	Matchear solo con espacios.
`[^\D]`	Matchear todo lo que no sean números.
`a\+`	Matchea con a+.
`b$`	Matchea un String que termina con b.
`a{2,3}`	Matchea aa o aaa.
`.*\.txt$`	Matchea Strings con extensión .txt.
`(a|b|c)a`	Matchea aa, ba o ca.
`[A-Z]`	Matchea cualquier mayúscula.
`[a-zA-Z]`	Matchea con cualquier minúscula o mayúscula.
`[a-zA-Z0-9]`	Matchea con cualquier minúscula, mayúscula o dígito.

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

```
([\w\.]+)@((?:[\w]+\.)+)([a-zA-Z]{2,4})
```

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

```
([\w\.]+)@((?:[\w]+\.)+)([a-zA-Z]{2,4})
```

- **1st Capturing Group** `([\w\.])`
  - **Match a single character present in the list below** `[\w\.]`

    `+` matches the previous token between `one` and `unlimited` times, as many times as possible, giving back as needed `(greedy)`

    `\w` matches any word character (equivalent to `[a-zA-Z0-9_]`)

    `\.` matches the character `.` literally (case sensitive)

https://regex101.com/

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

```
([\w\.]+)@((?:[\w]+\.)+)([a-zA-Z]{2,4})
```

- **1st Capturing Group** `([\w\.])`
  - **Match a single character present in the list below** `[\w\.]`
    `+` matches the previous token between `one` and `unlimited` times, as many times as possible, giving back as needed `(greedy)`
    `\w` matches any word character (equivalent to `[a-zA-Z0-9_]`)
    `\.` matches the character `.` literally (case sensitive)

`@` matches the character `@` literally (case sensitive)

https://regex101.com/

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

```
([\w\.]+)@((?:[\w]+\.)+)([a-zA-Z]{2,4})
```

- **2nd Capturing Group** `((?:[\w]\.)+)`
  - **Non-capturing group** `(?:[\w]\.)+`

    `+` matches the previous token between `one` and `unlimited` times, as many times as possible, giving back as needed `(greedy)`
    - **Match a single character present in the list below** `[\w]`

      `+` matches the previous token between `one` and `unlimited` times, as many times as possible, giving back as needed `(greedy)`

      `\w` matches any word character (equivalent to `[a-zA-Z0-9_]`)

    `\.` matches the character `.` literally (case sensitive)

https://regex101.com/

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

```
([\w\.]+)@((?:[\w]+\.)+)([a-zA-Z]{2,4})
```

▼ **3rd Capturing Group** `([a-zA-Z])`

▼ **Match a single character present in the list below** `[a-zA-Z]`

`{2,4}` matches the previous token between `2` and `4` times, as many times as possible, giving back as needed `(greedy)`

`a-z` matches a single character in the range between `a (index 97)` and `z (index 122)` (case sensitive)

`A-Z` matches a single character in the range between `A (index 65)` and `Z (index 90)` (case sensitive)

https://regex101.com/

Pre-procesamiento

Eliminar caracteres especiales → Qué son las Expresiones Regulares?

```
([\w\.]+)@((?:[\w]+\.)+)([a-zA-Z]{2,4})
```

probando.lala@hotmail.com

MATCH INFORMATION

Match 1	0-25	probando.lala@hotmail.com
Group 1	0-13	probando.lala
Group 2	14-22	hotmail.
Group 3	22-25	com

https://regex101.com/

Pre-procesamiento

Eliminar caracteres especiales → Acentos

- Usualmente en un conjunto de textos puede haber caracteres acentuados.

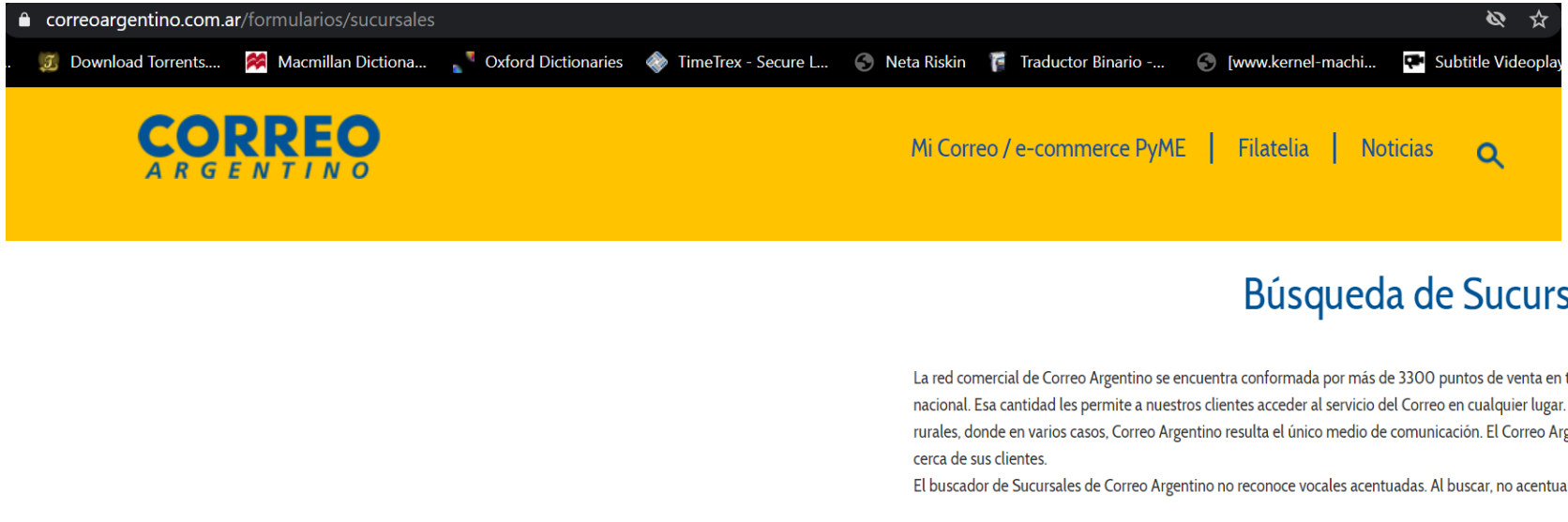
- Si los acentos no proveen información especial, deben ser estandarizados y convertidos en caracteres ASCII.
  - Por ejemplo, al procesar textos en Inglés.
  - Algún ejemplo en el que puedan ser relevantes?

Pre-procesamiento

Eliminar caracteres especiales → Acentos

- Usualmente en un conjunto de textos puede haber caracteres acentuados.

- Si los acentos no proveen información especial, deben ser estandarizados y convertidos en caracteres ASCII.
  - Por ejemplo, al procesar textos en Inglés.
  - Algún ejemplo en el que puedan ser relevantes?



Pre-procesamiento

Eliminar caracteres especiales → Acentos y otros caracteres “raros”

- Usualmente en un conjunto de textos puede haber caracteres acentuados.

- Si los acentos no proveen información especial, deben ser estandarizados y convertidos en caracteres ASCII.
  - Por ejemplo, al procesar textos en Inglés.
  - Algún ejemplo en el que puedan ser relevantes?

- Múltiple formas de normalización Unicode

(lograr una representación única de expresiones equivalentes):

- NFD.
- NFC.
- NFKD.
- NFKC.

NF: Normalization Form,
C: Composition, D: Decomposition, K: Compatibility

Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

Dos tipos de equivalencias entre caracteres

Equivalencia **canónica**

Equivalencia **de compatibilidad**

https://unicode.org/reports/tr15/#Norm_Forms

Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

Dos tipos de equivalencias entre caracteres

Equivalencia **canónica**

Equivalencia fundamental entre caracteres y secuencias que representan al mismo character abstracto. Cuando son impresos deben mantener la misma apariencia y comportamiento.

Equivalencia **de compatibilidad**

Es una equivalencia más débil entre caracteres y secuencias que representan al mismo carácter abstracto, pero que pueden tener diferencias visuales y de comportamiento.

Si la diferencia es solo estilística, podría ser compensada. Si la diferencia se relaciona con la semántica (por ejemplo, en contextos de fórmulas matemáticas), puede no ser recomendable.

https://unicode.org/reports/tr15/#Norm_Forms

Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

Dos tipos de equivalencias entre caracteres

Equivalencia **canónica**

Equivalencia fundamental entre caracteres y secuencias que representan al mismo character abstracto. Cuando son impresos deben mantener la misma apariencia y comportamiento.

Equivalencia **de compatibilidad**

Es una equivalencia más débil entre caracteres y secuencias que representan al mismo carácter abstracto, pero que pueden tener diferencias visuales y de comportamiento.

Si la diferencia es solo estilística, podría ser compensada. Si la diferencia se relaciona con la semántica (por ejemplo, en contextos de fórmulas matemáticas), puede no ser recomendable.

Super set
Cada equivalencia canónica es una equivalencia compatible

https://unicode.org/reports/tr15/#Norm_Forms

Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

Dos tipos de equivalencias entre caracteres

Equivalencia **canónica**

Secuencia combinada Ç ↔ C + ◌̧

Orden en las marcas q + ◌̇ + ◌̣ ↔ q + ◌̣ + ◌̇

Singleton Ω ↔ Ω

Equivalencia **de compatibilidad**

Diferencias de fuentes ℌ → H

ℍ → H

https://unicode.org/reports/tr15/#Norm_Forms

Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

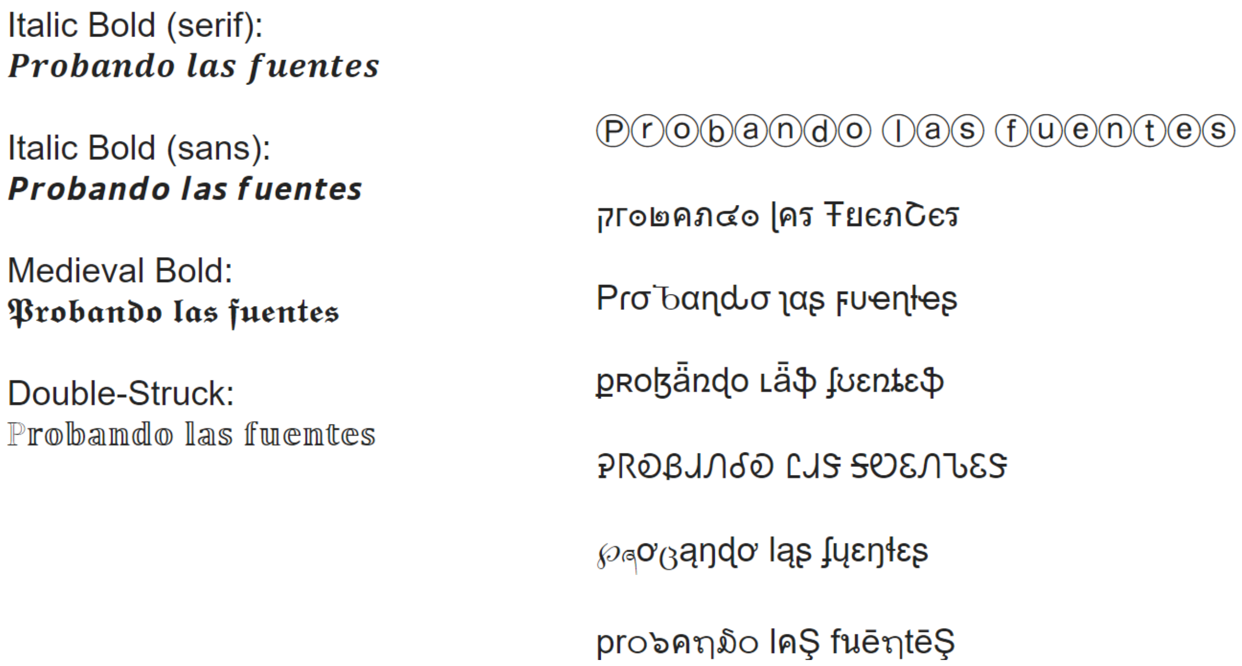
- La compatibilidad no está limitada solo a quitar acentos, sino también a diferencias derivadas del formato.
  - Variabilidad de fuentes.
  - Cursiva.
  - Caracteres con círculos.
  - Variación en ancho.
  - Variación de tamaño.
  - Rotación.
  - Super, supra índice.
  - Fracciones.

https://lingojam.com/BoldTextGenerator

Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

- La compatibilidad no está limitada solo a quitar acentos, sino también a diferencias derivadas del formato.
  - Variabilidad de fuentes.
  - Cursiva.
  - Caracteres con círculos.
  - Variación en ancho.
  - Variación de tamaño.
  - Rotación.
  - Super, supra índice.
  - Fracciones.



https://lingojam.com/BoldTextGenerator

Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

- Diferentes formas de normalización

		Descomposición	Composición
NFD	Normalization Form Decomposition	canónica	-
NFC	Normalization Form Composition	canónica	Si
NFKD	Normalization Form Compatibility Decomposition	compatible	
NFKC	Normalization Form Compatibility Composition	compatible	Si

- **Composición** combina las "marcas" con los char base. Múltiples elementos separados son combinados en uno, en la medida de lo posible.
- **Descomposición** divide chars compuestos en sus componentes simples.

https://unicode.org/reports/tr15/#Norm_Forms

Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

- Diferentes formas de normalización

		Descomposición	Composición
NFD	**Normalization Form Decomposition**	**canónica**	**-**

Separa los chars Unicode en sus componentes independientes.

Ç → C ̧

https://unicode.org/reports/tr15/#Norm_Forms

Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

- Diferentes formas de normalización

		Descomposición	Composición
NFD	Normalization Form Decomposition	canónica	-
NFC	**Normalization Form Composition**	**canónica**	**Si**

Primero descompone los chars como NFD, pero luego los vuelve a componer en un único char.

Ç → Ç → Ç

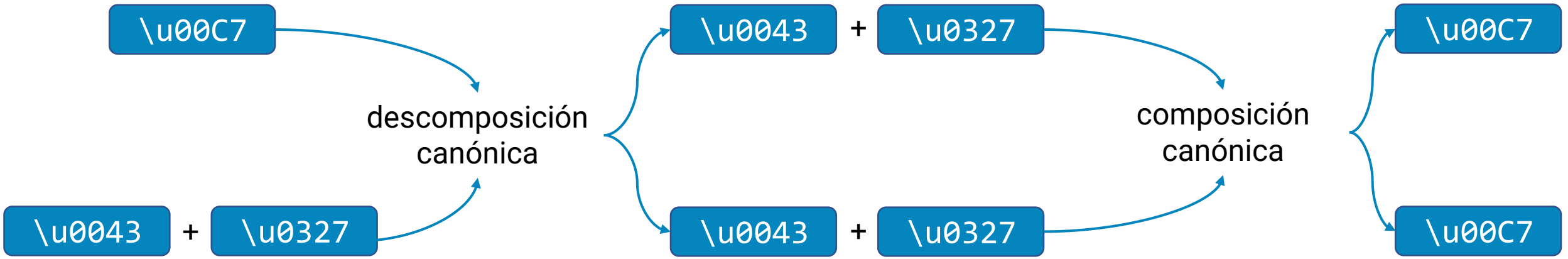
https://unicode.org/reports/tr15/#Norm_Forms

Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

- NFC

Ç → C̦ → Ç



Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

- Diferentes formas de normalización

		Descomposición	Composición
NFD	Normalization Form Decomposition	canónica	-
NFC	Normalization Form Composition	canónica	Si
NFKD	**Normalization Form Compatibility Decomposition**	**compatible**	

ℌ, → H,

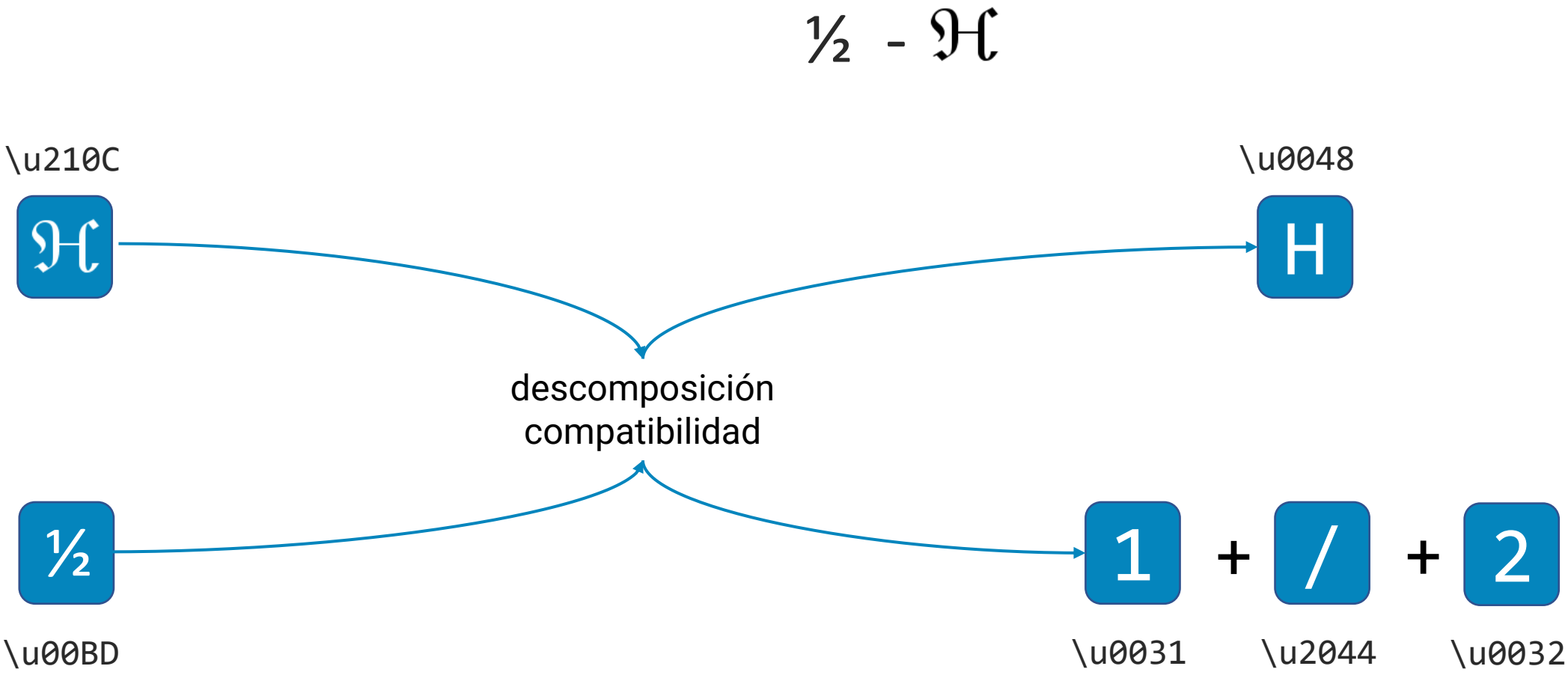
Separa los chars “fancy” o alternativos en sus componentes (en caso de que sea posible).
Los transforma en chars más ”normales”.

https://unicode.org/reports/tr15/#Norm_Forms

Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

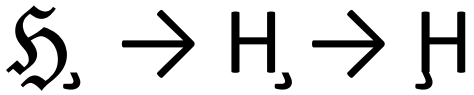
- NFKD



Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

- Diferentes formas de normalización



		Descomposición	Composición
NFD	Normalization Form Decomposition	canónica	-
NFC	Normalization Form Composition	canónica	Si
NFKD	Normalization Form Compatibility Decomposition	compatible	
NFKC	**Normalization Form Compatibility Composition**	**compatible**	**Si**

Son dos etapas, descomposición de compatibilidad seguido por una composición canónica.
Permite normalizar todas las variantes de un char a una única version común.

No se puede hacer una composición de compatibilidad dado que la equivalencia de compatibilidad funciona en un solo sentido.
Por ejemplo, hay una gran variedad de fuentes desde las cuales pudimos haber realizado la descomposición, como sabríamos a qué fuente deberíamos hacer la composición?

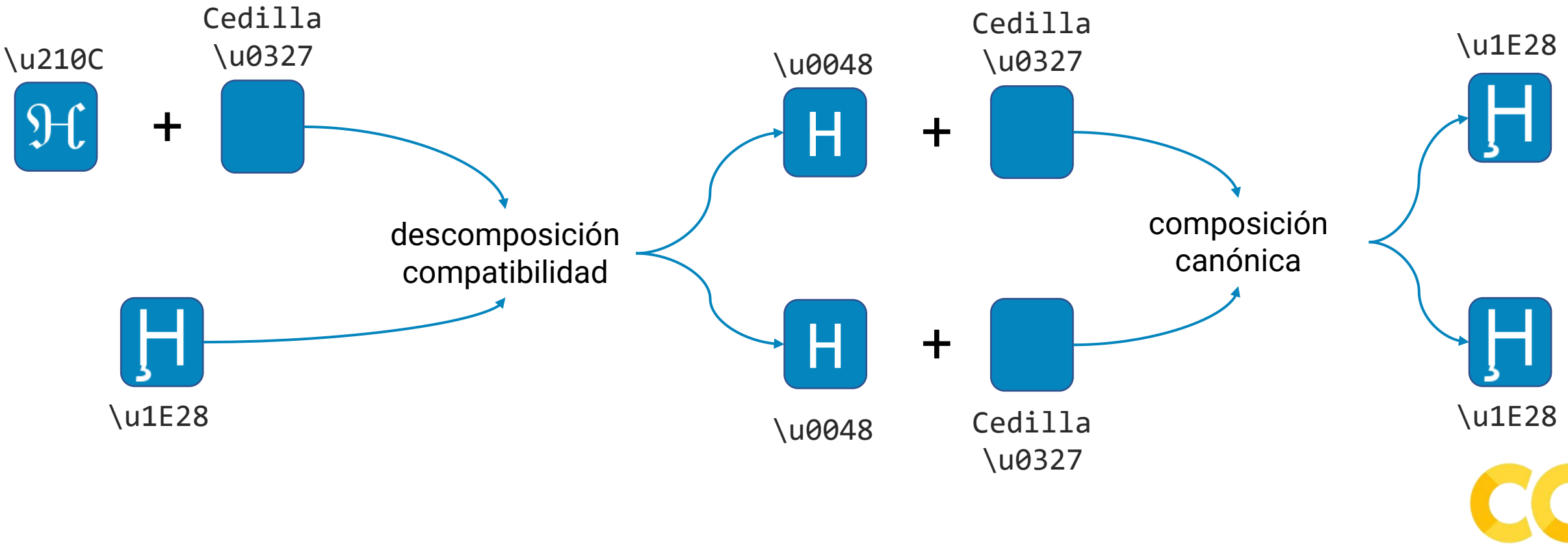
https://unicode.org/reports/tr15/#Norm_Forms

Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

- NFKC

ℌ̧ → Ḩ → Ḩ



Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

- Diferentes formas de normalización

		Descomposición	Composición
NFD	Normalization Form Decomposition	canónica	-
NFC	Normalization Form Composition	canónica	Si
NFKD	Normalization Form Compatibility Decomposition	compatible	
NFKC	Normalization Form Compatibility Composition	compatible	Si

No cambian la forma original

https://unicode.org/reports/tr15/#Norm_Forms

Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

- Diferentes formas de normalización

		Descomposición	Composición
NFD	Normalization Form Decomposition	canónica	-
NFC	Normalization Form Composition	canónica	Si
NFKD	Normalization Form Compatibility Decomposition	compatible	
NFKC	Normalization Form Compatibility Composition	compatible	Si

No cambian la longitud original

https://unicode.org/reports/tr15/#Norm_Forms

Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

- Diferentes formas de normalización

		Descomposición	Composición
NFD	Normalization Form Decomposition	canónica	-
NFC	Normalization Form Composition	canónica	Si
NFKD	Normalization Form Compatibility Decomposition	compatible	
NFKC	Normalization Form Compatibility Composition	compatible	Si

Se pueden generar errores de compatilibidad si dos aplicaciones normalizan de distinta forma!!

Errores de pérdidas de información que no son triviales de resolver dado que la normalización no puede ser reconstruída sin pérdida.

https://unicode.org/reports/tr15/#Norm_Forms

Pre-procesamiento

Eliminar caracteres especiales → Normalización Unicode

- Diferentes formas de normalización

		Descomposición	Composición
NFD	Normalization Form Decomposition	canónica	-
NFC	Normalization Form Composition	canónica	Si
NFKD	Normalization Form Compatibility Decomposition	compatible	
NFKC	Normalization Form Compatibility Composition	compatible	Si

La elección de cuál usar depende el sistema particular.

NFC suele ser la major opción para texto general dado que es más compatible con strings convertidos de encodings más viejos.

NFD y NFKD son útiles para procesamientos internos.

https://unicode.org/reports/tr15/#Norm_Forms

Pre-procesamiento

Segmentación/Tokenización de texto

- La tokenización se puede definir como el proceso de descomponer o dividir datos textuales en componentes más pequeños y significativos llamados **tokens**.

- Los tokens son componentes textuales independientes y mínimos que tienen una sintaxis y semántica definidas.

Pre-procesamiento

Segmentación/Tokenización de texto

- La tokenización se puede definir como el proceso de descomponer o dividir datos textuales en componentes más pequeños y significativos llamados **tokens**.

- Los tokens son componentes textuales independientes y mínimos que tienen una sintaxis y semántica definidas.

```
Mr. and Mrs. Dursley, of number
four, Privet Drive, were proud to
say that they were perfectly
normal, thank you very much. They
were the last people you'd expect
to be involved in anything
strange or mysterious, because
they just didn't hold with such
nonsense.
```

```
Mr. and Mrs. Dursley, of number
four, Privet Drive, were proud to
say that they were perfectly
normal, thank you very much.
```

```
Mr.
and
Mrs.
Dursley,
of
number
four,
Privet
Drive,
Were
proud
to
say
that
they
were
perfectly
normal,
thank
you
very
much.
```

Pre-procesamiento

Segmentación/Tokenización de texto

- La tokenización se puede definir como el proceso de descomponer o dividir datos textuales en componentes más pequeños y significativos llamados **tokens**.

- Los tokens son componentes textuales independientes y mínimos que tienen una sintaxis y semántica definidas.

- Un párrafo de texto o un documento de texto tiene varios componentes, incluidas oraciones, que se pueden dividir en cláusulas, frases y palabras.

- Se realiza la tokenización de **oraciones** y **palabras**, que se utilizan para dividir un documento de texto (o corpus) en oraciones y cada oración en palabras.

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de oraciones

- La **tokenización de oraciones** es el proceso de dividir un texto en oraciones que actúan como el primer nivel de tokens que comprende dicho texto.

NLTK
Natural Language Toolkit

- Hay varias formas de realizar la tokenización de oraciones.

- NLTK ofrece diferentes alternativas:

Sent_tokenize	Punkt	Regexp

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de oraciones

- La **tokenización de oraciones** es el proceso de dividir un texto en oraciones que actúan como el primer nivel de tokens que comprende dicho texto.

NLTK
Natural Language Toolkit

- Hay varias formas de realizar la tokenización de oraciones.

- NLTK ofrece diferentes alternativas:

Por defecto sirve para multiples idiomas.

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de oraciones

- La **tokenización de oraciones** es el proceso de dividir un texto en oraciones que actúan como el primer nivel de tokens que comprende dicho texto.

NLTK
Natural Language Toolkit

- Hay varias formas de realizar la tokenización de oraciones.

- NLTK ofrece diferentes alternativas:

- Divide el texto una lista de oraciones utilizando un modelo no supervisado que considera abreviaciones, collocations y palabras que comúnmente se encuentran al inicio de las oraciones.
    - El modelo requiere mucho texto de entrada (idealmente similar al texto que se tokenizará) para el entrenamiento.

Punkt

- Los espacios en blanco en el texto original se mantienen en la salida. Los signos de puntuación al final de las oraciones también son incluidos.

Por defecto sirve para multiples idiomas.

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de oraciones

- La **tokenización de oraciones** es el proceso de dividir un texto en oraciones que actúan como el primer nivel de tokens que comprende dicho texto.
- Hay varias formas de realizar la tokenización de oraciones.
- NLTK ofrece diferentes alternativas:

NLTK
Natural Language Toolkit

```
SENTENCE_TOKENS_PATTERN
= r'(?<!\w\.\w.)(?<![A-
Z][a-z]\.)(?<![A-Z]\.)
    (?<=\.|\?|\!)\s'
```

Sent_tokenize

Punkt

Regexp

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

- Una oración es una colección de palabras.
- La tokenización esencialmente divide una oración en su lista de palabras.

NLTK
Natural Language Toolkit

- La **tokenización de palabras** es realmente importante en muchos procesos, especialmente en la limpieza y normalización de textos donde las operaciones como la derivación y la lematización se aplican a cada palabra individual en función de sus respectivos stems y lemas.

- NLTK ofrece diferentes alternativas:

word_tokenize	TreebankWord Tokenizer	TokTok Tokenizer	Regexp

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

- Una oración es una colección de palabras.
- La tokenización esencialmente divide una oración en su lista de palabras.

NLTK
Natural Language Toolkit

- La **tokenización de palabras** es realmente importante en muchos procesos, especialmente en la limpieza y normalización de textos donde las operaciones como la derivación y la lematización se aplican a cada palabra individual en función de sus respectivos stems y lemas.

- NLTK ofrece diferentes alternativas:

word_tokenize

TreebankWord Tokenizer

TokTok Tokenizer

Regexp

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

- Una oración es una colección de palabras.
- La tokenización esencialmente divide una oración en su lista de palabras.

NLTK
Natural Language Toolkit

- La **tokenización de palabras** es realmente importante en muchos procesos, especialmente en la limpieza y normalización de textos donde las operaciones como la derivación y la lematización se aplican a cada palabra individual en función de sus respectivos stems y lemas.

- NLTK ofrece diferentes alternativas:

word_tokenize

TreebankWord Tokenizer

- Utiliza expresiones regulares.
- Asume que ya se realizó el tokenizado de oraciones.
- Sigue algunas reglas:
  - Separa los puntos al final de las operaciones.
  - Separa las comas y comillas simples seguidas por espacios.
  - Se separa la mayoría de los signos de puntuación.
  - Separa palabras y sus contracciones.

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

- Una oración es una colección de palabras.
- La tokenización esencialmente divide una oración en su lista de palabras.

NLTK
Natural Language Toolkit

- La **tokenización de palabras** es realmente importante en muchos procesos, especialmente en la limpieza y normalización de textos donde las operaciones como la derivación y la lematización se aplican a cada palabra individual en función de sus respectivos stems y lemas.

- NLTK ofrece diferentes alternativas:

  - Es uno de lo más nuevos.
  - Asume que la entrada es una oración por línea, con lo que solo el ultimo punto es tokenizado.
    - En caso de necesidad utiliza expresiones regulares para eliminar otros puntos.

TokTok Tokenizer

Regexp

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

- Una oración es una colección de palabras.
- La tokenización esencialmente divide una oración en su lista de palabras.

NLTK
Natural Language Toolkit

- La **tokenización de palabras** es realmente importante en muchos procesos, especialmente en la limpieza y normalización de textos donde las operaciones como la derivación y la lematización se aplican a cada palabra individual en función de sus respectivos stems y lemas.

- NLTK ofrece diferentes alternativas:

```
WordPunktTokenizer:
r'\w+|[^\w\s]+'
WhitespaceTokenizer: \t, \b, \n
```

Regexp

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

TreebankWord Tokenizer	TokTok Tokenizer	Stanford Tokenizer
Principalmente inglés. Se basa en expresiones regulares adaptadas a reglas de puntuación.	English, Persian, Russian, Czech, French, German, Vietnamese, …	PTBTokenizer funcionará bien sobre el texto codificado en Unicode que no requiere segmentación de palabras (como los sistemas de escritura que no ponen espacios entre las palabras) o reglas más “raras” específicas del lenguaje (como los sistemas de escritura que usan : o ? Como un carácter dentro de las palabras , etc.) Admite Unicode de plano multilingüe no básico, en particular, emojis. Proporciona opciones para japonés, chino y árabe.

Todos funcionan mejor en usos formales del Inglés!

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

TreebankWord Tokenizer	TokTok Tokenizer	Stanford Tokenizer
Principalmente inglés. Se basa en expresiones regulares adaptadas a reglas de puntuación.	English, Persian, Russian, Czech, French, German, Vietnamese, …	PTBTokenizer funcionará bien sobre el texto codificado en Unicode que no requiere segmentación de palabras (como los sistemas de escritura que no ponen espacios entre las palabras) o reglas más “raras” específicas del lenguaje (como los sistemas de escritura que usan : o ? Como un carácter dentro de las palabras , etc.) Admite Unicode de plano multilingüe no básico, en particular, emojis. Proporciona opciones para japonés, chino y árabe.

Todos funcionan mejor en usos formales del Inglés!
Para SMS o textos de social media necesitamos otras herramientas!

Por ejemplo:
https://github.com/leondz/twokenize

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

spaCy

1. El texto es dividido considerando los caracteres blancos.

2. Por cada substring, de izquierda a derecha, se hacen dos controles:

a. Coincide el String con una regla de excepción?

- Por ejemplo:
  - “don’t” no contiene un blanco, pero debe ser separado en “do” y “n’t”.
  - “U.K.” no debe separarse.

b. Puede extraerse un prefijo, sufijo o infijo?

- Por ejemplo, signos de puntuación (comas, puntos, comillas, hyphens).

Si hay una coincidencia se aplica la regla y se comienza con los Strings separados.

Esto le permite a spaCy dividir estructuras complejas como estructuras anidadas o combinaciones multiples signos de puntuación.

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

1. El texto es dividido considerando los caracteres blancos.

2. Por cada substring, de izquierda a derecha, se hacen dos controles:

a. Coincide el String con una regla de excepción?
- Por ejemplo:
  - “don’t” no contiene un blanco, pero debe ser separado en “do” y “n’t”.
  - “U.K.” no debe separarse.

b. Puede extraerse un prefijo, sufijo o infijo?
- Por ejemplo, signos de puntuación (comas, puntos, comillas, hyphens).

Si hay una coincidencia se aplica la regla y se comienza con los Strings separados.

Esto le permite a spaCy dividir estructuras complejas como estructuras anidadas o combinaciones multiples signos de puntuación.

spaCy

“Let’s go to N.Y.!”

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

spaCy

1. El texto es dividido considerando los caracteres blancos.

2. Por cada substring, de izquierda a derecha, se hacen dos controles:

a. Coincide el String con una regla de excepción?

- Por ejemplo:
  - “don’t” no contiene un blanco, pero debe ser separado en “do” y “n’t”.
  - “U.K.” no debe separarse.

b. Puede extraerse un prefijo, sufijo o infijo?

- Por ejemplo, signos de puntuación (comas, puntos, comillas, hyphens).

Si hay una coincidencia se aplica la regla y se comienza con los Strings separados.

Esto le permite a spaCy dividir estructuras complejas como estructuras anidadas o combinaciones multiples signos de puntuación.

“Let’s	go	to	N.Y.!”
“Let’s	go	to	N.Y.!”

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

1. El texto es dividido considerando los caracteres blancos.

2. Por cada substring, de izquierda a derecha, se hacen dos controles:

a. Coincide el String con una regla de excepción?
- Por ejemplo:
  - “don’t” no contiene un blanco, pero debe ser separado en “do” y “n’t”.
  - “U.K.” no debe separarse.

b. Puede extraerse un prefijo, sufijo o infijo?
- Por ejemplo, signos de puntuación (comas, puntos, comillas, hyphens).

Si hay una coincidencia se aplica la regla y se comienza con los Strings separados.

Esto le permite a spaCy dividir estructuras complejas como estructuras anidadas o combinaciones multiples signos de puntuación.

spaCy

“Let’s	go	to	N.Y.!”	
“Let’s	go	to	N.Y.!”	
“Let’s	go	to	N.Y.!”	Prefijo

Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

1. El texto es dividido considerando los caracteres blancos.

2. Por cada substring, de izquierda a derecha, se hacen dos controles:

a. Coincide el String con una regla de excepción?

- Por ejemplo:
  - “don’t” no contiene un blanco, pero debe ser separado en “do” y “n’t”.
  - “U.K.” no debe separarse.

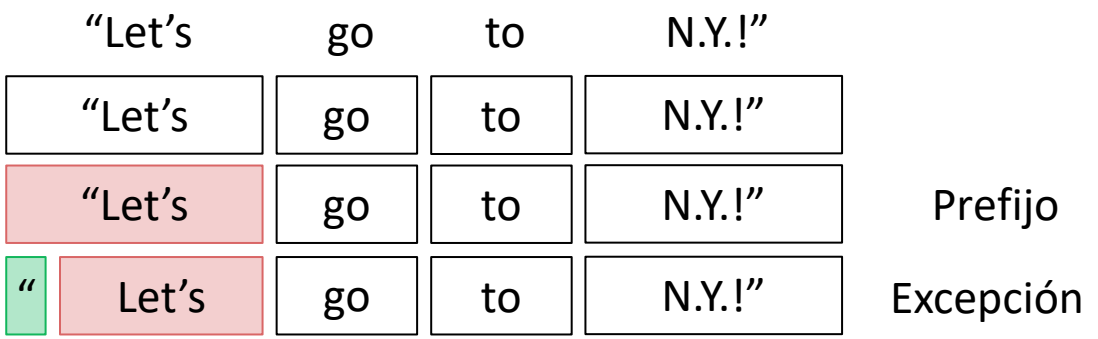
b. Puede extraerse un prefijo, sufijo o infijo?

- Por ejemplo, signos de puntuación (comas, puntos, comillas, hyphens).

Si hay una coincidencia se aplica la regla y se comienza con los Strings separados.

Esto le permite a spaCy dividir estructuras complejas como estructuras anidadas o combinaciones multiples signos de puntuación.

spaCy



Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

1. El texto es dividido considerando los caracteres blancos.

2. Por cada substring, de izquierda a derecha, se hacen dos controles:

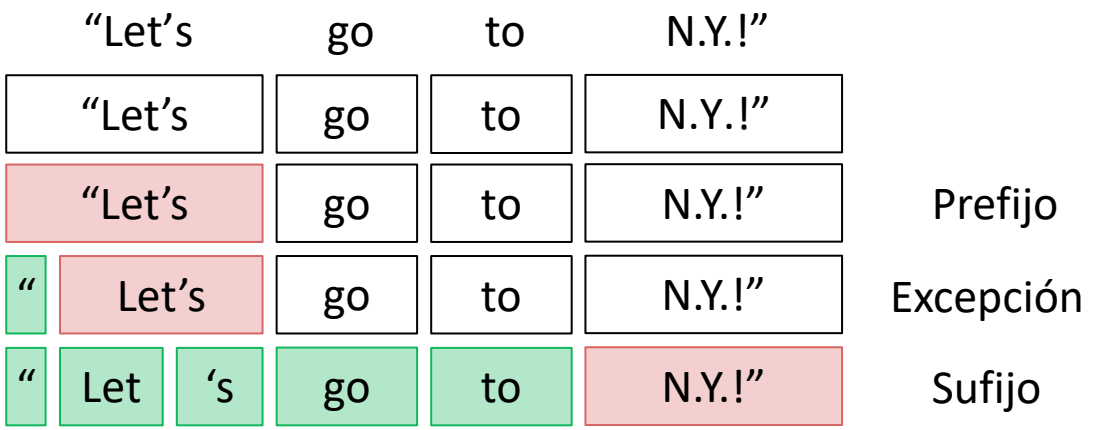
a. Coincide el String con una regla de excepción?
- Por ejemplo:
  - “don’t” no contiene un blanco, pero debe ser separado en “do” y “n’t”.
  - “U.K.” no debe separarse.

b. Puede extraerse un prefijo, sufijo o infijo?
- Por ejemplo, signos de puntuación (comas, puntos, comillas, hyphens).

Si hay una coincidencia se aplica la regla y se comienza con los Strings separados.

Esto le permite a spaCy dividir estructuras complejas como estructuras anidadas o combinaciones multiples signos de puntuación.

spaCy



Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

1. El texto es dividido considerando los caracteres blancos.

2. Por cada substring, de izquierda a derecha, se hacen dos controles:

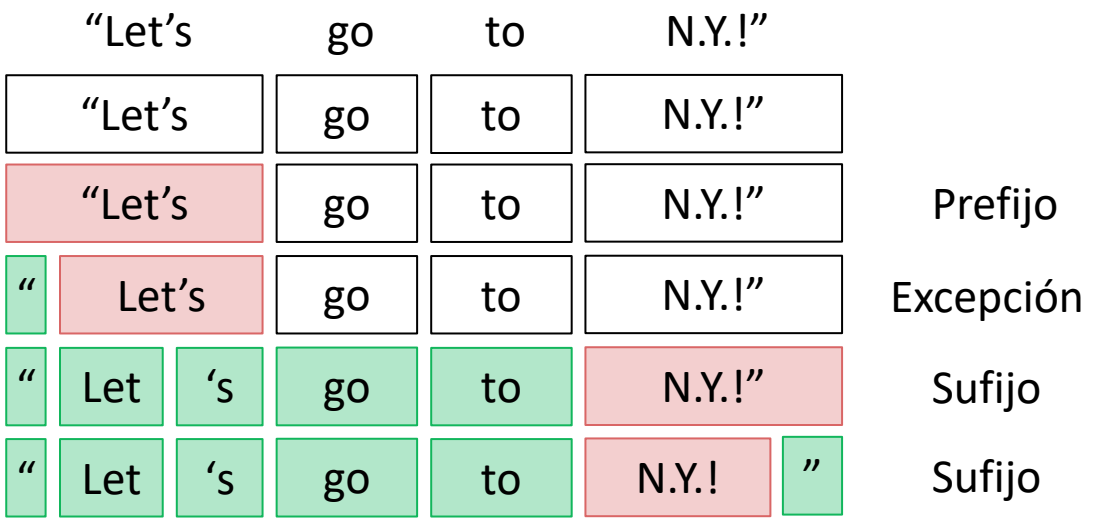
a. Coincide el String con una regla de excepción?
- Por ejemplo:
  - “don’t” no contiene un blanco, pero debe ser separado en “do” y “n’t”.
  - “U.K.” no debe separarse.

b. Puede extraerse un prefijo, sufijo o infijo?
- Por ejemplo, signos de puntuación (comas, puntos, comillas, hyphens).

Si hay una coincidencia se aplica la regla y se comienza con los Strings separados.

Esto le permite a spaCy dividir estructuras complejas como estructuras anidadas o combinaciones multiples signos de puntuación.

spaCy



Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

1. El texto es dividido considerando los caracteres blancos.

2. Por cada substring, de izquierda a derecha, se hacen dos controles:

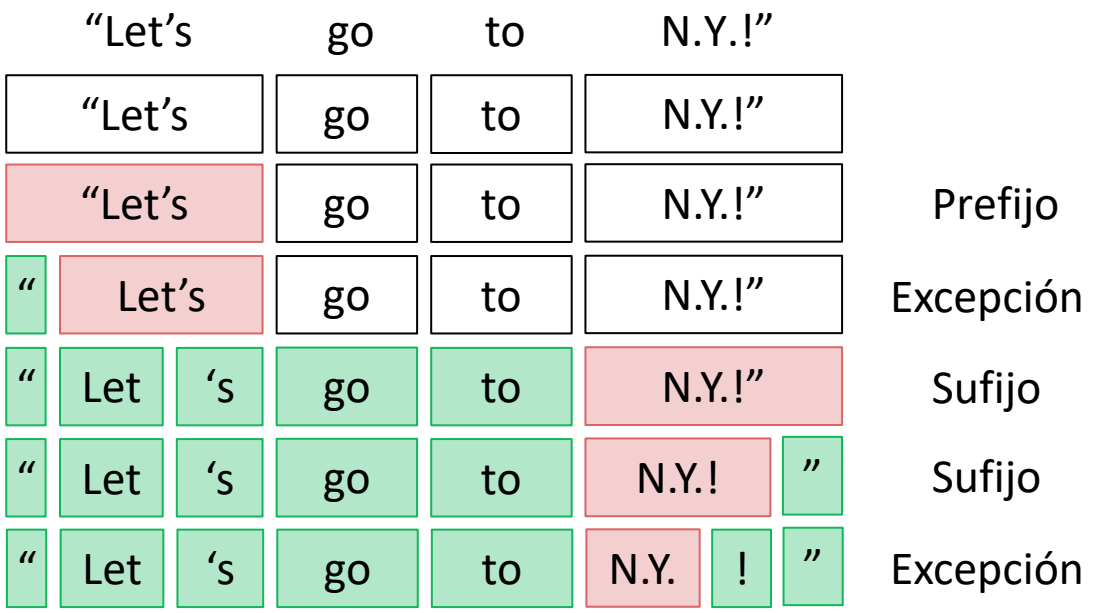
a. Coincide el String con una regla de excepción?
- Por ejemplo:
  - “don’t” no contiene un blanco, pero debe ser separado en “do” y “n’t”.
  - “U.K.” no debe separarse.

b. Puede extraerse un prefijo, sufijo o infijo?
- Por ejemplo, signos de puntuación (comas, puntos, comillas, hyphens).

Si hay una coincidencia se aplica la regla y se comienza con los Strings separados.

Esto le permite a spaCy dividir estructuras complejas como estructuras anidadas o combinaciones multiples signos de puntuación.

spaCy



Pre-procesamiento

Segmentación/Tokenización de texto → Tokenización de palabras

1. El texto es dividido considerando los caracteres blancos.

2. Por cada substring, de izquierda a derecha, se hacen dos controles:

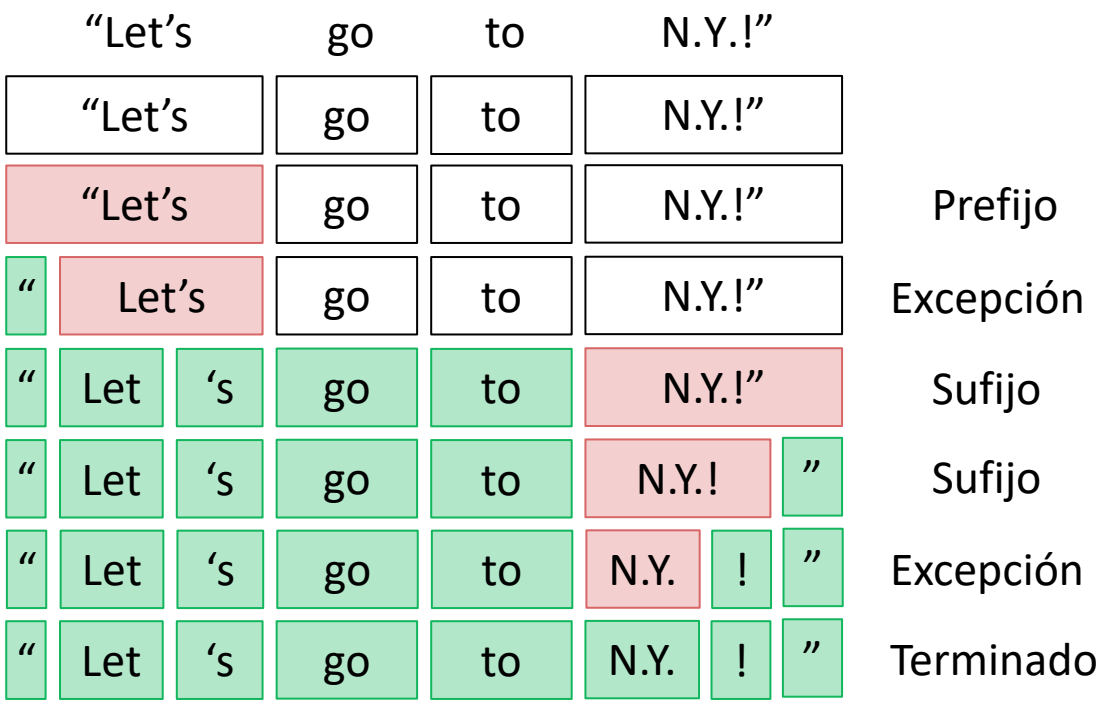
a. Coincide el String con una regla de excepción?
- Por ejemplo:
  - “don’t” no contiene un blanco, pero debe ser separado en “do” y “n’t”.
  - “U.K.” no debe separarse.

b. Puede extraerse un prefijo, sufijo o infijo?
- Por ejemplo, signos de puntuación (comas, puntos, comillas, hyphens).

Si hay una coincidencia se aplica la regla y se comienza con los Strings separados.

Esto le permite a spaCy dividir estructuras complejas como estructuras anidadas o combinaciones multiples signos de puntuación.

spaCy



Pre-procesamiento

Expansión de contracciones

- Las contracciones son versiones abreviadas de palabras o sílabas.
- Se crean eliminando letras y sonidos específicos.
- A menudo se evitan cuando se escribe por escrito, pero se usan de manera bastante extensa en la comunicación informal.
- Existen varias formas de contracciones y están vinculadas al tipo de verbos auxiliares, negaciones, ….

Pre-procesamiento

Expansión de contracciones

- Las contracciones son versiones abreviadas de palabras o sílabas.
- Se crean eliminando letras y sonidos específicos.
- A menudo se evitan cuando se escribe por escrito, pero se usan de manera bastante extensa en la comunicación informal.
- Existen varias formas de contracciones y están vinculadas al tipo de verbos auxiliares, negaciones, ….
- Plantean un problema a NLP dado que:
  - Agregan un carácter extra en el diccionario.
  - Generan excepciones al tokenizar y estandarizar.
- Idealmente, puede tener un mapeo adecuado para las contracciones y sus expansiones correspondientes y luego usarlo para expandir todas las contracciones en un texto.

Pre-procesamiento

Expansión de contracciones

- La mayoría de las alternativas utilizan diccionarios de contracciones.
  - No considera el contexto.
  - Resuelve ambigüedades considerando cual es la expansion con mayor probabilidad.

Pre-procesamiento

Expansión de contracciones

- La mayoría de las alternativas utilizan diccionarios de contracciones.
  - No considera el contexto.
  - Resuelve ambigüedades considerando cual es la expansion con mayor probabilidad.

I'd like to know how I'd done that!

Pre-procesamiento

Expansión de contracciones

- La mayoría de las alternativas utilizan diccionarios de contracciones.
  - No considera el contexto.
  - Resuelve ambigüedades considerando cual es la expansion con mayor probabilidad.

I'd like to know how I'd done that!

Si considerásemos la forma más común:

I **would** like to know how I **would** done that!

Pre-procesamiento

Expansión de contracciones

- La mayoría de las alternativas utilizan diccionarios de contracciones.
  - No considera el contexto.
  - Resuelve ambigüedades considerando cual es la expansion con mayor probabilidad.

I'd like to know how I'd done that!

Si considerásemos la forma más común:

I **would** like to know how I **would** done that!

Sin embargo, la forma correcta sería:

I **would** like to know how I **had** done that!

Pre-procesamiento

Expansión de contracciones

pycontractions

Para la expansión, utiliza un enfoque basado en embeddings de tres pasos:

1. Contracciones simples que solo matchean con una única regla son expandidas.

2. Para aquellas contracciones que matchean con más de una regla:
   a. Se reemplaza la contracción por todas las posibles reglas.
   b. Cada nuevo texto es pasado por un corrector gramatical y se calcula una distancia entre los nuevos textos y el original.

3. Se ordenan las hipótesis considerando la cantidad de errores (menor a mayor) y la distancia entre el texto original y las hipótesis (menor a mayor distancia). Se selecciona la primera del ranking.

- La distancia es medida considerando "Word Mover's Distance (WMD)", la cual mide la distancia entre dos documentos como la mínima distancia que los embeddings de cada uno de ellos debe moverse para alcanzar los embeddings del otro documento.
- Debido a que la diferencia entre cada una de las hipótesis de reemplazado, la hipótesis más cercana a la del texto original será aquella con la menor distancia entre los embeddings correspondientes a la contracción y la hipótesis.
- Puede o no asumir que para un mismo texto las contracciones se expanderán siempre de la misma manera.

Pre-procesamiento

Expansión de contracciones

pycontractions

Para la expansión, utiliza un enfoque basado en embeddings de tres pasos:

1. Contracciones simples que solo matchean con una única regla son expandidas.

Puede fallar!

2. Para aquellas contracciones que matchean con más de una regla:
    a. Se reemplaza la contracción por todas las posibles reglas.
    b. Cada nuevo texto es pasado por un corrector gramatical y se calcula una distancia entre los nuevos textos y el original.

3. Se ordenan las hipótesis considerando la cantidad de errores (menor a mayor) y la distancia entre el texto original y las hipótesis (menor a mayor distancia). Se selecciona la primera del ranking.

- La distancia es medida considerando "Word Mover's Distance (WMD)", la cual mide la distancia entre dos documentos como la mínima distancia que los embeddings de cada uno de ellos debe moverse para alcanzar los embeddings del otro documento.
- Debido a que la diferencia entre cada una de las hipótesis de reemplazado, la hipótesis más cercana a la del texto original será aquella con la menor distancia entre los embeddings correspondientes a la contracción y la hipótesis.
- Puede o no asumir que para un mismo texto las contracciones se expanderán siempre de la misma manera.

Pre-procesamiento

Expansión de contracciones

pycontractions

We ain’t all the same.

Hipótesis	WMD	Errores Gramaticales
We have not all the same	0.6680542214210519	0
We are not all the same	0.7372250927409768	0
We has not all the same	0.7223834627019157	1
We am not all the same	0.8113022453418426	1
We is not all the same	0.6954222661000212	2

Pre-procesamiento

Expansión de contracciones

pycontractions

We ain’t all the same.

Hipótesis	WMD	Errores Gramaticales
We have not all the same	0.6680542214210519	0
We are not all the same	0.7372250927409768	0
We has not all the same	0.7223834627019157	1
We am not all the same	0.8113022453418426	1
We is not all the same	0.6954222661000212	2

Pre-procesamiento

Expansión de contracciones

pycontractions

We ain’t all the same.

Hipótesis	WMD	Errores Gramaticales
We have not all the same	0.6680542214210519	0
We are not all the same	0.7372250927409768	0
We has not all the same	0.7223834627019157	1
We am not all the same	0.8113022453418426	1
We is not all the same	0.6954222661000212	2

- Los errores se deben a cómo fueron generados los embeddings utilizados.
- En este caso,
  - Qué tiempos verbales/verbos se encuentran más relacioandos al “we” y al “not”?

Pre-procesamiento

Eliminar Stopwords

- Las **stopwords** son palabras que tienen poca o ninguna importancia y generalmente se eliminan del texto cuando se procesan para retener las palabras que tienen la máxima importancia y contexto.

- No existe una lista universal o exhaustiva de palabras clave y, a menudo, cada dominio o idioma tiene su propio conjunto de palabras clave!
  - NLTK, spaCy y Stanford NLP proporcionan sus propias listas (en inglés).

- Una cosa importante para recordar es que las **negaciones** están incluidas en la lista.
  - Sin embargo, a menudo **deben preservarse** para que el significado actual de la oración no se pierda en las aplicaciones!
    - Por ejemplo, en el caso de sentiment analysis.

Pre-procesamiento

Eliminar Stopwords

```
a
about
above
after
again
against
all
am
an
and
any
are
aren't
as
at
be
because
been
before
Being
below
between
both
but
by
can
can't
cannot
could
couldn't
did
didn't
do
does
doesn't
doing
don't
down
during
each
few
for
from
further
had
hadn't
has
hasn't
have
haven't
having
he
he'd
he'll
he's
her
here
here's
hers
herself
him
himself
his
how
how's
i
i'd
i'll
i'm
i've
if
in
into
is
isn't
it
it's
its
itself
let's
me
more
most
mustn't
my
myself
no
nor
not
of
off
on
once
only
or
other
ought
our
ours
ourselves
out
over
own
same
shan't
she
she'd
she'll
she's
should
shouldn't
so
some
such
than
that
that's
the
their
theirs
them
themselves
then
there
there's
these
they
they'd
they'll
they're
they've
this
those
through
to
too
under
until
up
very
was
wasn't
we
we'd
we'll
we're
we've
were
weren't
what
what's
when
when's
where
where's
which
while
who
who's
whom
why
why's
with
won't
would
wouldn't
you
you'd
you'll
you're
you've
your
yours
yourself
yourselves
###
return
arent
cant
couldnt
didnt
doesnt
dont
hadnt
hasnt
havent
hes
heres
hows
im
isnt
its
lets
mustnt
shant
shes
shouldnt
thats
theres
theyll
theyre
theyve
wasnt
were
werent
whats
whens
wheres
whos
whys
wont
wouldnt
youd
youll
youre
youve
```

https://github.com/stanfordnlp/CoreNLP/blob/master/data/edu/stanford/nlp/patterns/surface/stopwords.txt

Pre-procesamiento

Eliminar Stopwords

a
al
algo
algunas
algunos
ante
antes
como
con
contra
cual
cuando
de
del
desde
donde
durante
e
El
él
ella
ellas
ellos
en
entre
era
erais
eran
eras
eres
es
esa
esas
ese
eso
esos
esta
estaba
estabais
estaban
estabas
estad
estada
estadas
estado
estados
estamos
estando
estar
estaremos
estará
estarán
estarás
estaré
estaréis
estaría
estaríais
estaríamos
estarían
estarías
estas
este
estemos
esto
estos
estoy
estuve
estuviera
estuvierais
estuvieran
estuvieras
estuvieron
estuviese
estuvieseis
estuviesen
estuvieses
estuvimos
estuviste
estuvisteis
estuviéramos
estuviésemos
estuvo
está
estábamos
estáis
están
estás
esté
estéis
estén
estés
fue
fuera
fuerais
fueran
fueras
fueron
fuese
fueseis
fuesen
fueses
fui
fuimos
fuiste
fuisteis
fuéramos
fuésemos
ha
habida
habidas
habido
habidos
habiendo
habremos
habrá
habrán
habrás
habré
habréis
habría
habríais
habríamos
habrían
habrías
habéis
había
habíais
habíamos
habían
habías
han
has
hasta
hay
haya
hayamos
hayan
hayas
hayáis
he
hemos
hube
hubiera
hubierais
hubieran
hubieras
hubieron
hubiese
hubieseis
hubiesen
hubieses
hubimos
hubiste
hubisteis
hubiéramos
hubiésemos
hubo
la
las
le
les
lo
los
me
mi
mis
mucho
muchos
muy
más
mí
mía
mías
mío
míos
nada
ni
no
nos
nosotras
nosotros
nuestra
nuestras
nuestro
nuestros
o
os
otra
otras
otro
otros
para
pero
poco
por
porque
que
quien
quienes
qué
se
sea
seamos
sean
seas
seremos
será
serán
serás
seré
seréis
sería
seríais
seríamos
serían
serías
seáis
sido
siendo
sin
sobre
sois
somos
son
soy
su
sus
suya
suyas
suyo
suyos
sí
también
tanto
te
tendremos
tendrá
tendrán
tendrás
tendré
tendréis
tendría
tendríais
tendríamos
tendrían
tendrías
tened
tenemos
tenga
tengamos
tengan
tengas
tengo
tengáis
tenida
tenidas
tenido
tenidos
teniendo
tenéis
tenía
teníais
teníamos
tenían
tenías
ti
tiene
tienen
tienes
todo
todos
tu
tus
tuve
tuviera
tuvierais
tuvieran
tuvieras
tuvieron
tuviese
tuvieseis
tuviesen

https://github.com/Alir3z4/stop-words/blob/master/spanish.txt

Pre-procesamiento

Eliminar Stopwords

Eliminar	Quizás NO eliminar
• Clasificación de texto. • Filtrado de spam. • Identificación de idioma. • Clasificación género. • Generación de tags. Tareas que se benefician de las **ideas generales** o **conceptos.**	• Traducción. • Modelado del lenguaje. • Resumen. • Question-answering. • Análisis de sentimientos. Tareas que necesitan de las **peculiaridades** de cada una de las **palabras involucradas**.

Pre-procesamiento

Corrección de texto

- Uno de los principales desafíos que enfrenta el procesamiento de texto es la presencia de palabras incorrectas en el texto.

- Dos posibilidades:
  - Palabras que tienen errores ortográficos.
  - Palabras con varias letras repetidas que no contribuyen mucho a su significado general.

- El objetivo principal es estandarizar diferentes formas de estas palabras a la forma correcta para no terminar perdiendo información vital de diferentes tokens en el texto.

Pre-procesamiento

Corrección de texto

- Uno de los principales desafíos que enfrenta el procesamiento de texto es la presencia de palabras incorrectas en el texto.

- Dos posibilidades:
  - Palabras que tienen errores ortográficos.
  - Palabras con varias letras repetidas que no contribuyen mucho a su significado general.

- El objetivo principal es estandarizar diferentes formas de estas palabras a la forma correcta para no terminar perdiendo información vital de diferentes tokens en el texto.

finally

Pre-procesamiento

Corrección de texto

- Uno de los principales desafíos que enfrenta el procesamiento de texto es la presencia de palabras incorrectas en el texto.

- Dos posibilidades:
  - Palabras que tienen errores ortográficos.
  - Palabras con varias letras repetidas que no contribuyen mucho a su significado general.

- El objetivo principal es estandarizar diferentes formas de estas palabras a la forma correcta para no terminar perdiendo información vital de diferentes tokens en el texto.

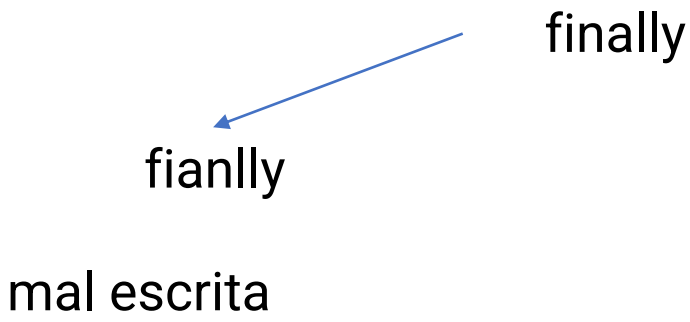
Pre-procesamiento

Corrección de texto

- Uno de los principales desafíos que enfrenta el procesamiento de texto es la presencia de palabras incorrectas en el texto.

- Dos posibilidades:
  - Palabras que tienen errores ortográficos.
  - Palabras con varias letras repetidas que no contribuyen mucho a su significado general.

- El objetivo principal es estandarizar diferentes formas de estas palabras a la forma correcta para no terminar perdiendo información vital de diferentes tokens en el texto.



Pre-procesamiento

Corrección de texto

- Uno de los principales desafíos que enfrenta el procesamiento de texto es la presencia de palabras incorrectas en el texto.

- Dos posibilidades:
  - Palabras que tienen errores ortográficos.
  - Palabras con varias letras repetidas que no contribuyen mucho a su significado general.

- El objetivo principal es estandarizar diferentes formas de estas palabras a la forma correcta para no terminar perdiendo información vital de diferentes tokens en el texto.

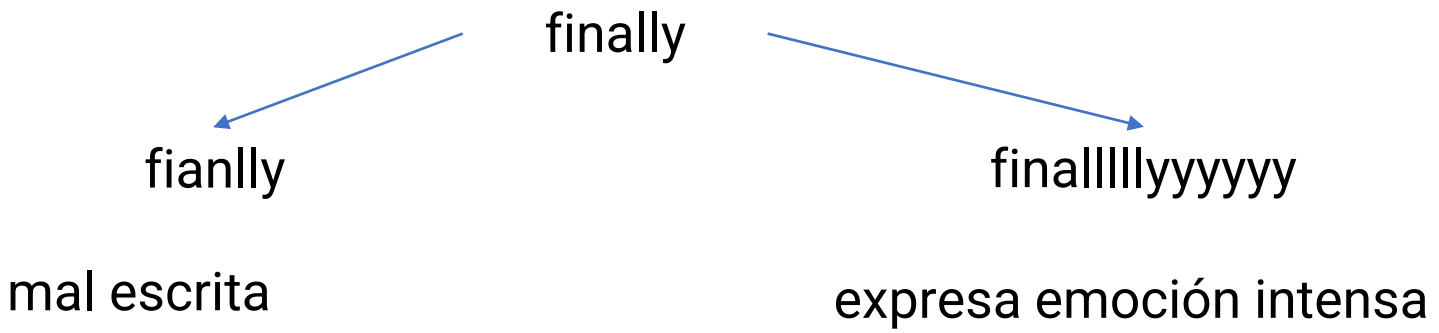
Pre-procesamiento

Corrección de texto

- Uno de los principales desafíos que enfrenta el procesamiento de texto es la presencia de palabras incorrectas en el texto.

- Dos posibilidades:
  - Palabras que tienen errores ortográficos.
  - Palabras con varias letras repetidas que no contribuyen mucho a su significado general.

- El objetivo principal es estandarizar diferentes formas de estas palabras a la forma correcta para no terminar perdiendo información vital de diferentes tokens en el texto.



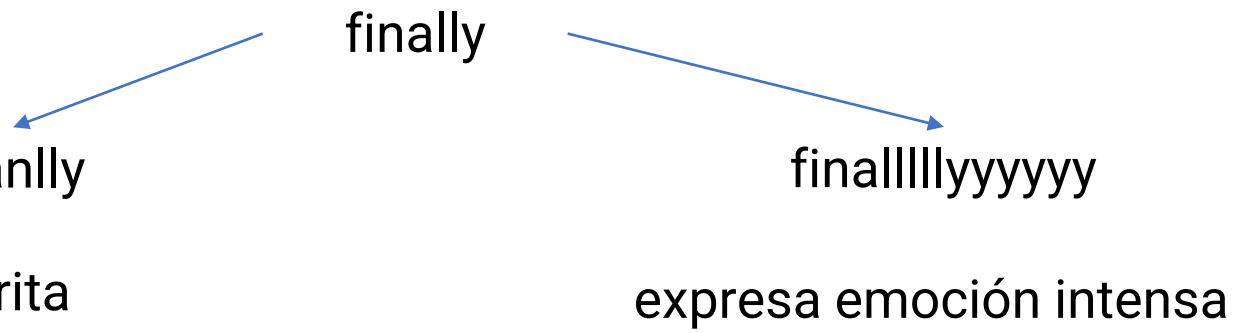
Pre-procesamiento

Corrección de texto

- Uno de los principales desafíos que enfrenta el procesamiento de texto es la presencia de palabras incorrectas en el texto.

- Dos posibilidades:
  - Palabras que tienen errores ortográficos.
  - Palabras con varias letras repetidas que no contribuyen mucho a su significado general.

- El objetivo principal es estandarizar diferentes formas de estas palabras a la forma correcta para no terminar perdiendo información vital de diferentes tokens en el texto.



Ojo!! Para algunas tareas puede ser relevante

Pre-procesamiento

Corrección de texto: Caracteres repetidos

- Palabras que a menudo contienen varios caracteres repetidos que podrían deberse a deletreos incorrectos, lenguaje de jerga o incluso a personas que desean expresar emociones fuertes.

- Combinación de sintaxis y semántica para corregir estas palabras.

- **Sintaxis**: identificar los caracteres repetidos en una palabra usando un patrón de expresiones regulares y luego usar una sustitución para eliminar los caracteres uno por uno.
  - `r '(\ w *) (\ w) \ 2 (\ w *) ’` → identificar caracteres que ocurren dos veces entre otros caracteres en la palabra.
  - Eliminar uno de los caracteres repetidos.
  - Repetir este proceso hasta que no queden caracteres repetidos.

Pre-procesamiento

Corrección de texto: Caracteres repetidos

- Palabras que a menudo contienen varios caracteres repetidos que podrían deberse a deletreos incorrectos, lenguaje de jerga o incluso a personas que desean expresar emociones fuertes.

- Combinación de sintaxis y semántica para corregir estas palabras.

- **Sintaxis**: identificar los caracteres repetidos en una palabra usando un patrón de expresiones regulares y luego usar una sustitución para eliminar los caracteres uno por uno.
  - `r '(\ w *) (\ w) \ 2 (\ w *) ’` → identificar caracteres que ocurren dos veces entre otros caracteres en la palabra.
  - Eliminar uno de los caracteres repetidos.
  - Repetir este proceso hasta que no queden caracteres repetidos.

```
Start Word: finalllyyy
```

Pre-procesamiento

Corrección de texto: Caracteres repetidos

- Palabras que a menudo contienen varios caracteres repetidos que podrían deberse a deletreos incorrectos, lenguaje de jerga o incluso a personas que desean expresar emociones fuertes.

- Combinación de sintaxis y semántica para corregir estas palabras.

- **Sintaxis**: identificar los caracteres repetidos en una palabra usando un patrón de expresiones regulares y luego usar una sustitución para eliminar los caracteres uno por uno.
  - `r '(\ w *) (\ w) \ 2 (\ w *) ’` → identificar caracteres que ocurren dos veces entre otros caracteres en la palabra.
  - Eliminar uno de los caracteres repetidos.
  - Repetir este proceso hasta que no queden caracteres repetidos.

```
Start Word: finalllyyy
Step: 1 Word: finalllyy
```

Pre-procesamiento

Corrección de texto: Caracteres repetidos

- Palabras que a menudo contienen varios caracteres repetidos que podrían deberse a deletreos incorrectos, lenguaje de jerga o incluso a personas que desean expresar emociones fuertes.

- Combinación de sintaxis y semántica para corregir estas palabras.

- **Sintaxis**: identificar los caracteres repetidos en una palabra usando un patrón de expresiones regulares y luego usar una sustitución para eliminar los caracteres uno por uno.
  - `r '(\ w *) (\ w) \ 2 (\ w *) ’` → identificar caracteres que ocurren dos veces entre otros caracteres en la palabra.
  - Eliminar uno de los caracteres repetidos.
  - Repetir este proceso hasta que no queden caracteres repetidos.

```
Start Word: finalllyyy
Step: 1 Word: finalllyy
Step: 2 Word: finallly
```

Pre-procesamiento

Corrección de texto: Caracteres repetidos

- Palabras que a menudo contienen varios caracteres repetidos que podrían deberse a deletreos incorrectos, lenguaje de jerga o incluso a personas que desean expresar emociones fuertes.

- Combinación de sintaxis y semántica para corregir estas palabras.

- **Sintaxis**: identificar los caracteres repetidos en una palabra usando un patrón de expresiones regulares y luego usar una sustitución para eliminar los caracteres uno por uno.
  - `r '(\ w *) (\ w) \ 2 (\ w *) ’` → identificar caracteres que ocurren dos veces entre otros caracteres en la palabra.
  - Eliminar uno de los caracteres repetidos.
  - Repetir este proceso hasta que no queden caracteres repetidos.

```
Start Word: finalllyyy
Step: 1 Word: finalllyy
Step: 2 Word: finallly
Step: 3 Word: finally
Step: 4 Word: finaly
Final word: finaly
```

Pre-procesamiento

Corrección de texto: Caracteres repetidos

- Palabras que a menudo contienen varios caracteres repetidos que podrían deberse a deletreos incorrectos, lenguaje de jerga o incluso a personas que desean expresar emociones fuertes.

- Combinación de sintaxis y semántica para corregir estas palabras.

- **Sintaxis**: identificar los caracteres repetidos en una palabra usando un patrón de expresiones regulares y luego usar una sustitución para eliminar los caracteres uno por uno.
  - `r '(\ w *) (\ w) \ 2 (\ w *) ’` → identificar caracteres que ocurren dos veces entre otros caracteres en la palabra.
  - Eliminar uno de los caracteres repetidos.
  - Repetir este proceso hasta que no queden caracteres repetidos.

- **Semántica**: la palabra obtenida podría no ser correcta.
  - Usar algún corpus o diccionario para verificar las palabras válidas en cada etapa y terminar el ciclo una vez que se obtiene.

Pre-procesamiento

Corrección de texto: Corrección de spelling

- Problema con la ortografía incorrecta que se produce debido a un error humano.
  - Lo que intentan solucionar los diccionarios predictivos.

- El objetivo final es que los tokens tengan la ortografía correcta.

- Basado en corpus y frecuencia de palabras.
  - Ej. Corpus de Gutenberg (corpus de 443 millones de palabras), Wiktionary y el Corpus Nacional Británico.
- Algoritmo de Norvig → Director de investigación de Google. Precisión del 80% ~ 90%.
  - Generar un conjunto de palabras candidatas que estén cerca de la palabra a corregir y seleccionar la palabra más probable de este conjunto como la palabra correcta.
  - Edit distance para medir la semejanza (cuántos cambios tengo que hacer en la palabra para obtener la similar?)
  - Usar un corpus de palabras correctamente escritas para identificar cual es la palabra “estadísticamente correcta” en relación a su frecuencia.

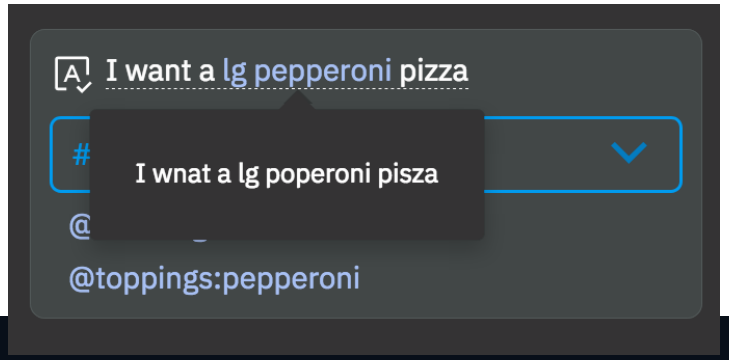
- No es provisto por todas las bibliotecas.

https://norvig.com/spell-correct.html

Pre-procesamiento

Corrección de texto: Corrección de spelling

- Según IBM, el 20% de los mensajes contienen errores tipográficos.
- Dependiendo de la importancia relativa que cada palabra tenga en el modelo entrenado, existe la posibilidad de que una falta ortográfica genere problemas de entendimiento.
- El asistente de Watson proporciona corrección ortográfica adaptativa.
- Analiza la oración completa del usuario y los textos de entrenamiento para determinar cada corrección.
  - La misma falta de ortografía puede corregirse de manera diferente según el contexto de la oración en la que se encuentra.
  - También significa que las palabras incluidas en el entrenamiento del sistema de diálogo no serán corregidas.



https://www.ibm.com/cloud/watson-assistant/

Análisis Léxico

- Se ocupa del estudio a nivel de las palabras con respecto a su significado léxico.
- Este nivel de procesamiento lingüístico utiliza el lexicon de un idioma, que es una colección de lexemas individuales.
- Un **lexema** es una unidad básica de significado léxico; que es una unidad abstracta de análisis morfológico que representa el conjunto de formas o "sentidos" tomados por un solo morfema.

Stemming

Lemmatization

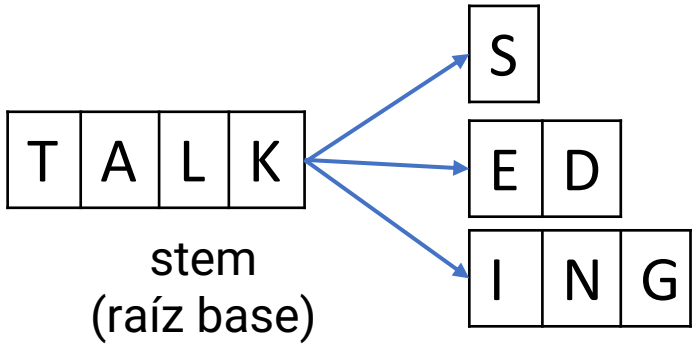
Análisis Léxico

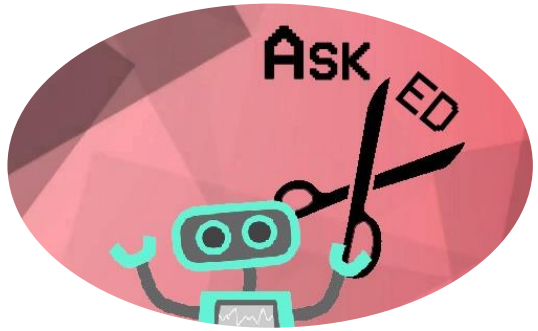
Stemming

- Los morfemas consisten en unidades que son raíces y afijos.
- Los afijos son unidades como prefijos, sufijos, etc., que se agregan a los formas base de las palabras para cambiar su significado o crear una palabra completamente nueva.

stem + afijo = palabra → inflection

palabra – afijo = stem → stemming





- El stemming ayuda a estandarizar las palabras en su raíz base, independientemente de sus inflexiones.
  - Útil para la clasificación, agrupamiento o incluso recuperación de información.

Stemming

Porter

Snowball

Lancaster

Stemming

Porter | Snowball | Lancaster

- **Porter.**
  - Uno de los primeros algoritmos.
  - Soporta idioma inglés.
  - Busca eliminar las terminaciones comunes de las palabras para que puedan resolverse a una forma común.
  - Garantiza la reproducibilidad.
  - Poco agresivo.
  - Es un buen algoritmo básico, pero no se recomienda usarlo para ninguna aplicación “real”.

Porter | Snowball | Lancaster

- **Snowball.**
  - Mejora sobre Porter.
  - Es un poco más rápido que Porter.
  - Más agresivo que Porter.
  - Danish, Dutch, English, Finnish, French, German, Hungarian, Italian, Norwegian, Portuguese, Romanian, Russian, Spanish, Swedish.

Porter | Snowball | Lancaster

- **Lancaster.**
  - Algoritmo muy agresivo y rápido.
  - A diferencia de Porter y Snowball, las representaciones derivadas no son intuitivas.
  - Los stems pueden ser "confusos".
    - No es recomendable si se quieren stems reconocibles o interpretables.
  - En NLTK, se le pueden agregar más reglas.

Análisis Léxico

Stemming

- Generalmente basados en heurísticas.
  - Dos problemas.

Análisis Léxico

Stemming

- Generalmente basados en heurísticas.
  - Dos problemas.

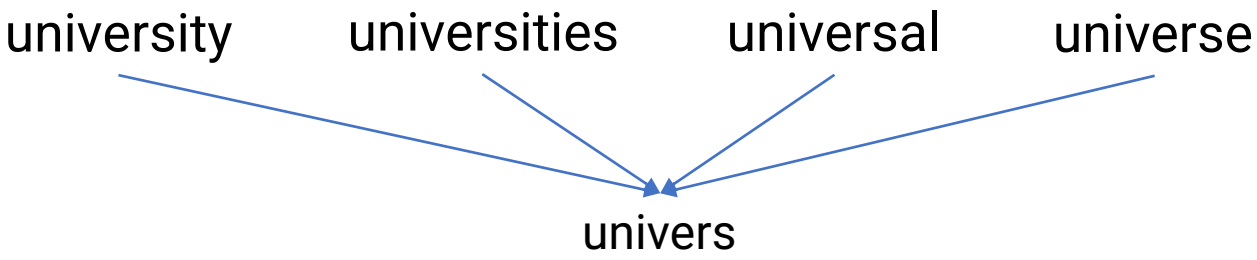
- **Overstemming.**
  - Reduce de sobra.
  - Stems sin sentido.
  - Palabras de origen distinto se reducen al mismo stem.
    - Probablemente no deban.

Análisis Léxico

Stemming

- Generalmente basados en heurísticas.
  - Dos problemas.

- **Overstemming.**
  - Reduce de sobra.
  - Stems sin sentido.
  - Palabras de origen distinto se reducen al mismo stem.
    - Probablemente no deban.

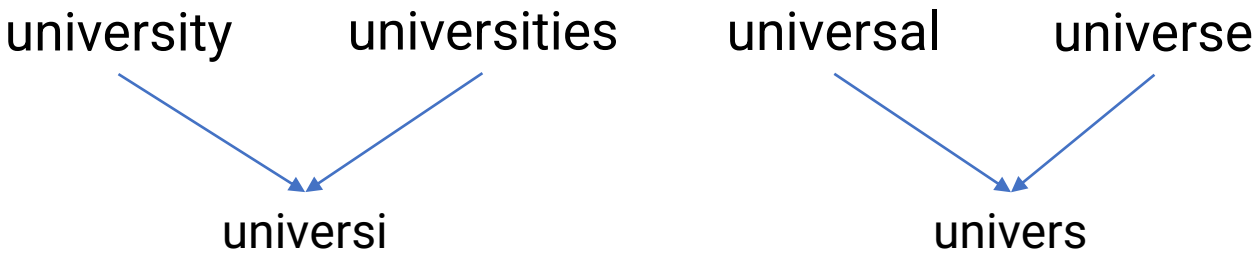


Análisis Léxico

Stemming

- Generalmente basados en heurísticas.
  - Dos problemas.

- **Overstemming.**
  - Reduce de sobra.
  - Stems sin sentido.
  - Palabras de origen distinto se reducen al mismo stem.
    - Probablemente no deban.



Debe tenerse cuidado que las reglas que se introduzcan para asegurar estas distinciones no ocasionen más problemas!!

Análisis Léxico

Stemming

- Generalmente basados en heurísticas.
  - Dos problemas.

- **Overstemming.**
  - Palabras de origen distinto se reducen al mismo stem.
    - Probablemente no deban.

- **Understemming.**
  - Se tienen múltiples palabras que son la forma de otra.
  - Las palabras son mapeadas a diferentes stems, cuando probablemente deban ser mapeados al mismo.
    - Podría seguir reduciendo pero no lo hace.

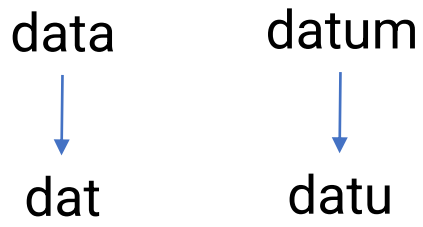
Análisis Léxico

Stemming

- Generalmente basados en heurísticas.
  - Dos problemas.

- **Overstemming.**
  - Palabras de origen distinto se reducen al mismo stem.
    - Probablemente no deban.

- **Understemming.**
    - Podría seguir reduciendo pero no lo hace.



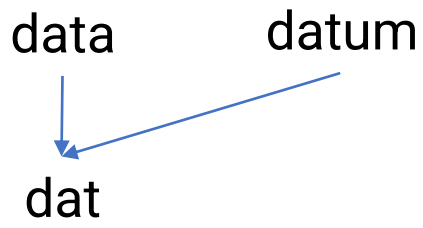
Análisis Léxico

Stemming

- Generalmente basados en heurísticas.
  - Dos problemas.

- **Overstemming.**
  - Palabras de origen distinto se reducen al mismo stem.
    - Probablemente no deban.

- **Understemming.**
    - Podría seguir reduciendo pero no lo hace.



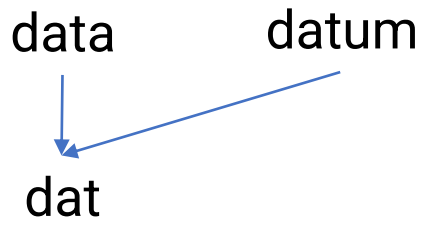
Análisis Léxico

Stemming

- Generalmente basados en heurísticas.
  - Dos problemas.

- **Overstemming.**
  - Palabras de origen distinto se reducen al mismo stem.
    - Probablemente no deban.

- **Understemming.**
    - Podría seguir reduciendo pero no lo hace.



Qué hacemos con la palabra “date”?

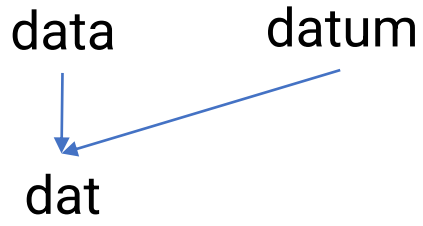
Análisis Léxico

Stemming

- Generalmente basados en heurísticas.
  - Dos problemas.

- **Overstemming.**
  - Palabras de origen distinto se reducen al mismo stem.
    - Probablemente no deban.

- **Understemming.**
    - Podría seguir reduciendo pero no lo hace.



Qué hacemos con la palabra “date”?

Hay que tener cuidado con cómo se resuelve.
Pensar si alguna regla general o si se está generando una nueva regla por un caso muy particular.

Análisis Léxico

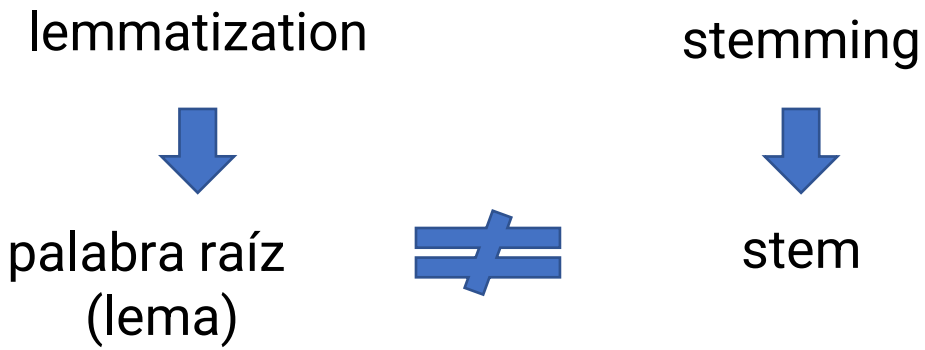
Lemmatización

- Similar a stemming, elimina los afijos de las palabras para llegar a su forma base.

Análisis Léxico

Lemmatización

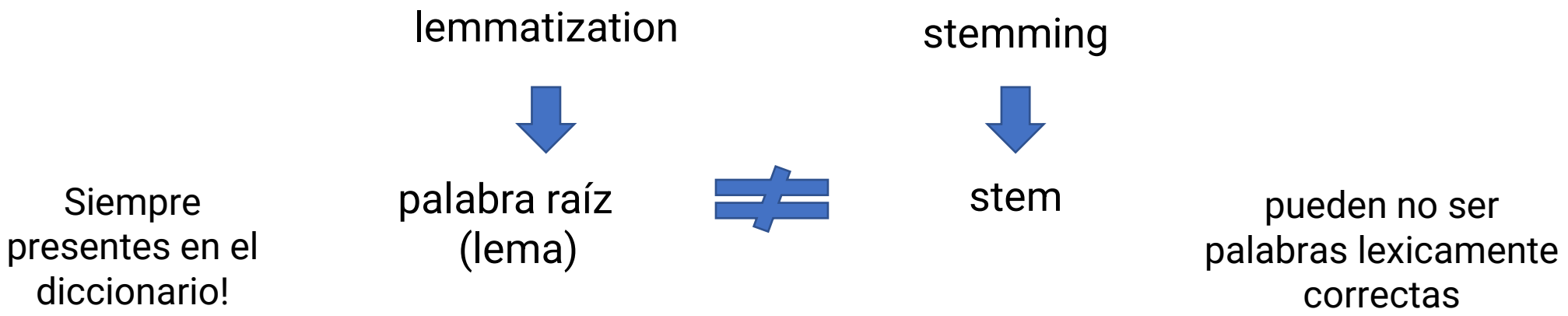
- Similar a stemming, elimina los afijos de las palabras para llegar a su forma base.



Análisis Léxico

Lemmatización

- Similar a stemming, elimina los afijos de las palabras para llegar a su forma base.



- La lematización es considerablemente más lenta que el stemming.
  - Implica un paso adicional en el que se forma la forma raíz o el lema eliminando el afijo de la palabra si y solo si el lema está presente en el diccionario.

Análisis Léxico

Lemmatización

- NLTK utiliza WordNet y la sintaxis y semántica de la palabra como parte del discurso y el contexto para obtener la palabra raíz o lema.
    - La part-of-speech es extremadamente importante porque si eso está mal, la lematización no será efectiva.
    - Puede funcionar sin part-of-speech, pero podría no ser efectivo.

- SpaCy es más fácil de usar.
    - Realiza el etiquetado de part-of-speech y lematización juntos.

Pre-procesamiento & Análisis Léxico

Qué incluir en el pipeline?

- El objetivo de construir el pipeline de pre-procesamiento es:
  - Extraer toda la información relevante.
  - Descartar toda la información irrelevante;
  - Representar el texto de forma más eficiente y compacta.

- No todas las tareas necesitan el mismo nivel de preprocesamiento.

Tarea	Qué necesitamos?	Pipeline
Topic modeling	Palabras con un significado semántico y asociación temática fuerte.	Tokenization, lemmatization, stopword removal, punctuation removal
Name Entity Recognition	Todas las palabras, incluyendo el rol que las mismas cumplen la oración.	tokenization, part of speech tagging
Sentiment analysis	Todas las palabras, incluidas las negaciones. No importan los tiempos verbales o forma de las palabras.	tokenization, lemmatization, sentence segmentation, punctuation removal

Pre-procesamiento & Análisis Léxico

Qué incluir en el pipeline?

Si o si

- Eliminar ruido
- Lowercasing
  - Puede depender de la tarea en algunas ocasiones.

Se podría hacer

- Normalización simple
  - Standarizar palabras léxicamente similares.
  - Corrección de spelling

Depende de la tarea

- Normalización avanzada
  - Qué hacer con palabras que no existen en el diccionario o corpus?
- Stopword removal
- Stemming / lemmatization

Pre-procesamiento & Análisis Léxico

Qué incluir en el pipeline?

	Particular a un dominio – Textos ruidosos	Texto general – Textos bien escritos
Muchos datos	• Pre-procesamiento moderado. • Podría ser útil enriquecer el texto.	• Pre-procesamiento liviano. • No es crítico enriquecer el texto.

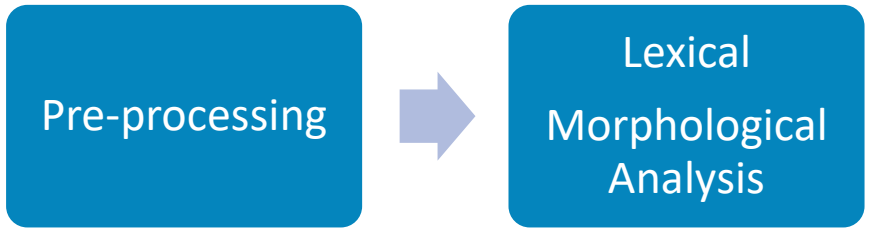
Cantidades limitadas	• Pre-procesamiento fuerte.
	• Importante enriquecer el texto.
	  • Agregar fuentes adicionales.
	  • Palabras similares.
		• Pre-procesamiento moderado.
		• Podría ser útil enriquecer el texto.

Resumen!

NLP es la rama de las ciencias de la computación que se enfoca en el desarrollo de sistemas que permita a las computadoras comunicarse con las personas.

El objetivo final es **leer**, **descifrar**, **comprender** y **dar sentido** a los lenguajes humanos de una forma que sea valiosa.

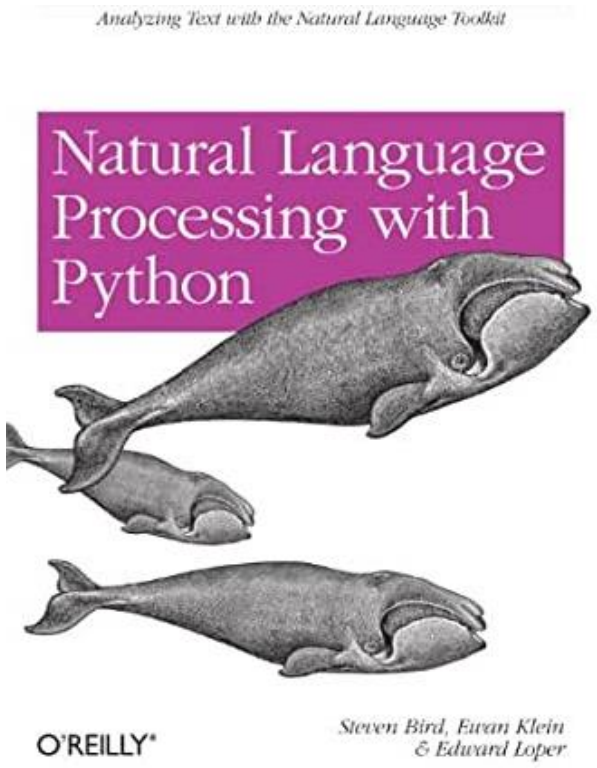
- Aplicaciones esenciales de NLP: búsquedas, recuperación de información, corrección de grammar, resumen, traducción, ….

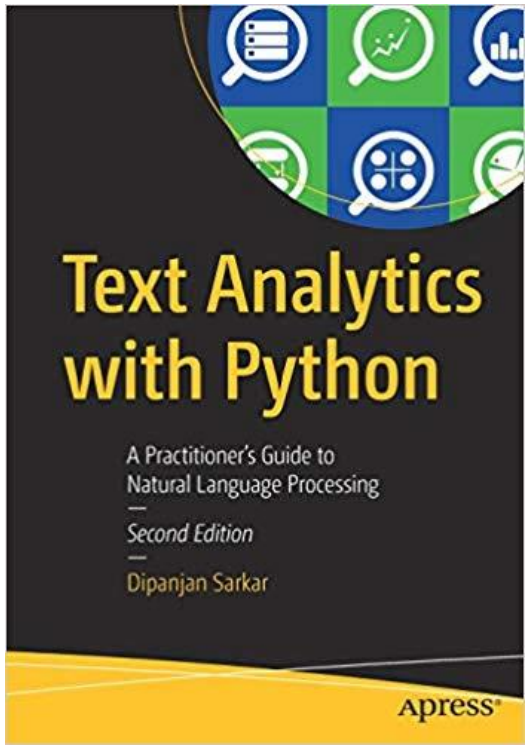


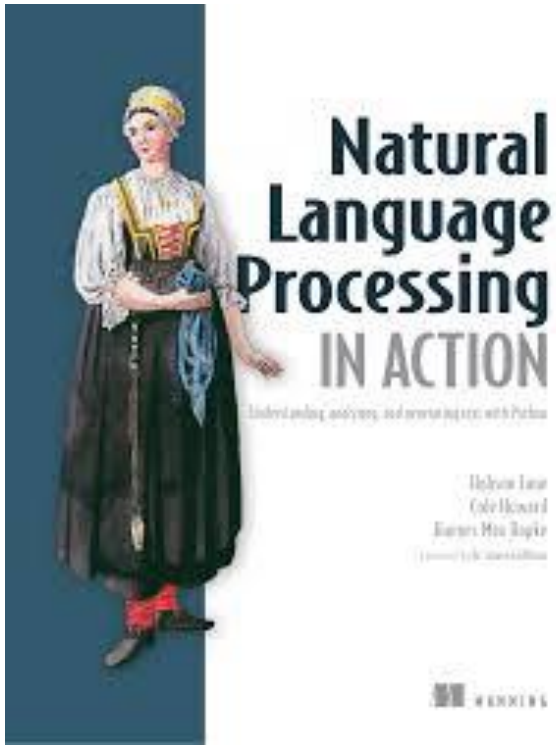
- **Pre-procesamiento.** Serie de pasos para limpiar y estandarizar el texto en una forma que pueda ser consumida por otros sistemas.
- **Análisis Léxico**. Se ocupa del estudio a nivel de las palabras con respecto a su significado léxico.

- Qué aplicar depende de la tarea a realizar!!

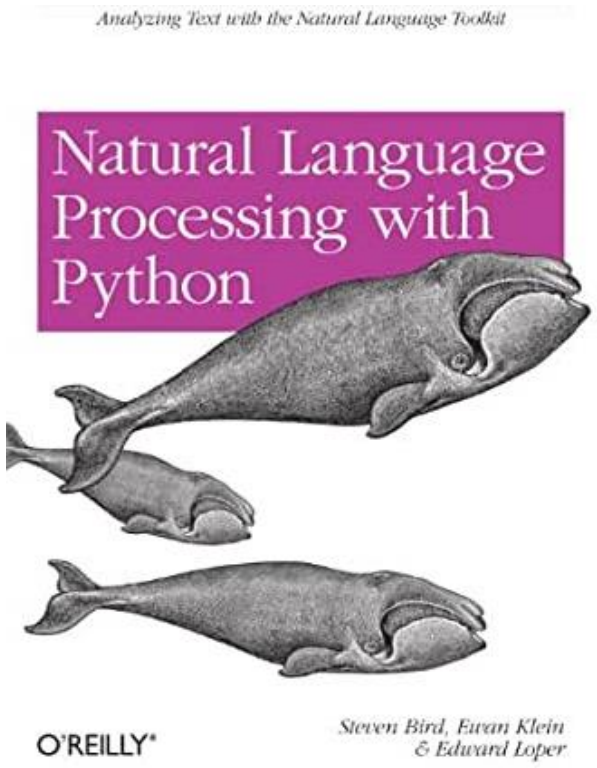
Lecturas sugeridas



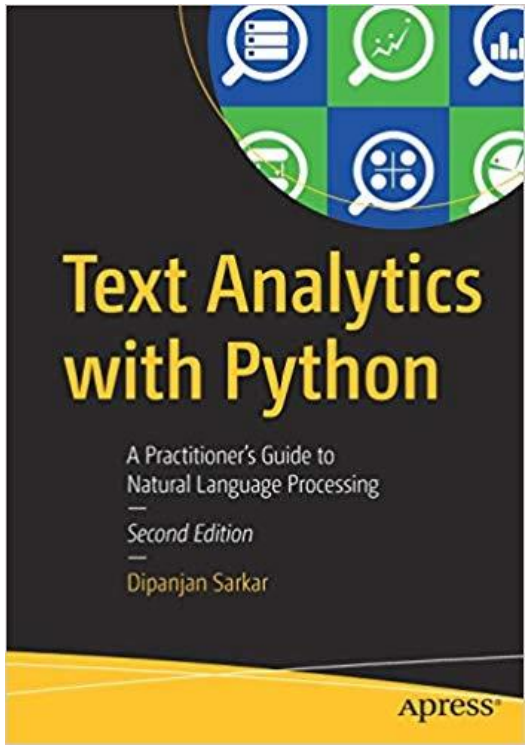




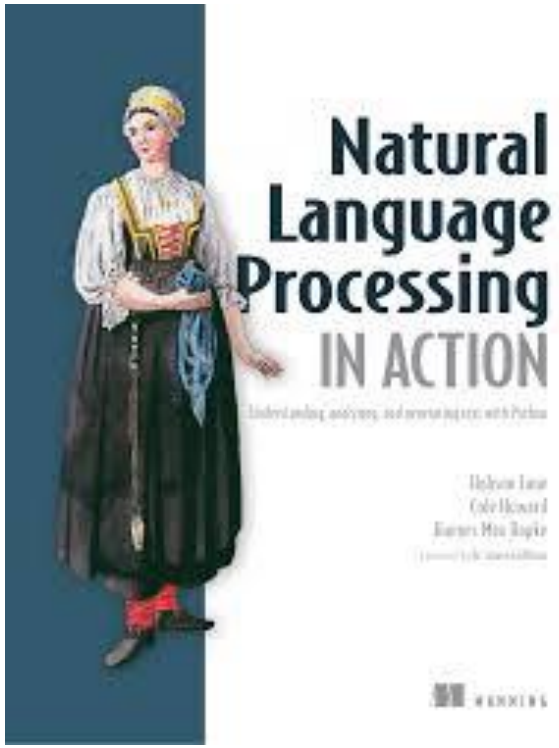
Lecturas sugeridas



Capítulo 3. “Processing Raw Text”



Capítulo 3. “Processing and Understanding Text”



Capítulo 2. “Build your vocabulary (word tokenization)”

Procesamiento de Lenguaje Natural

Pre-procesamiento – Análisis Léxico