

U3 :: Procesamiento del Lenguaje Natural

1. Preprocesamiento de texto

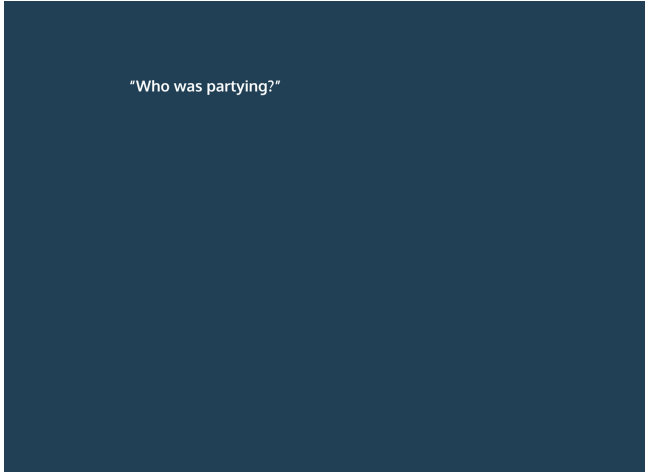
1.1. Introducción

El *preprocesamiento de texto* es un enfoque para limpiar y preparar datos de texto para su uso en un contexto específico. Los desarrolladores lo utilizan en casi todas las tareas de procesamiento del lenguaje natural (PLN), incluido el software de reconocimiento de voz, la búsqueda en motores de búsqueda y el entrenamiento de modelos de aprendizaje automático. Es un paso esencial porque los datos de texto pueden variar. Desde su formato (sitio web, mensaje de texto, reconocimiento de voz) hasta las personas que crean el texto (idioma, dialecto), hay muchas cosas que pueden introducir ruido en sus datos.

El objetivo final de limpiar y preparar datos de texto es reducir el texto a sólo las palabras que necesita para sus objetivos de PNL.

En esta lección, veremos diferentes técnicas para preparar datos de tipo texto. Si bien esta lista no es exhaustiva, cubriremos algunos enfoques comunes para limpiar y procesar datos de texto. Incluyen:

- Uso de bibliotecas Regex y NLTK
- Eliminación de caracteres y formato innecesarios
- *Tokenización*: dividir cadenas de varias palabras en componentes más pequeños
- *Normalización*: término general para procesar datos, que incluye derivación y lematización

A large, solid dark blue square graphic that occupies the lower half of the page.

"Who was partying?"

1.2. Eliminación de ruido

El limpiado de texto es una técnica que los desarrolladores utilizan en una variedad de dominios. Dependiendo del objetivo del proyecto y de dónde se obtengan los datos, es posible que desees eliminar información no deseada, como:

- Puntuación y acentos
- Caracteres especiales
- Dígitos numéricos
- Espacios en blanco inicial, final y vertical
- Formato HTML

El tipo de ruido que se necesita eliminar en el texto generalmente depende de su fuente. Por ejemplo, puede acceder a los datos a través de la API de Twitter, extrayendo una página web o un software de reconocimiento de voz. Afortunadamente, puede usar el método **.sub()** del módulo "re" de expresiones regulares de Python para la mayoría de casos que entrañan eliminación de ruido.

El método **.sub()** tiene tres argumentos obligatorios:

1. patrón: una expresión regular que se busca en la cadena de entrada. Debe haber una *r* antes de la cadena para indicar que es una cadena sin formato, que trata las barras invertidas como caracteres literales.
2. texto de reemplazo: texto que reemplaza todas las coincidencias en la cadena de
3. entrada input: la cadena de entrada que será editada por el método **.sub()**

El método devuelve una cadena con todas las instancias del patrón reemplazadas por texto de reemplazo. Veamos algunos ejemplos del uso de este método para eliminar y reemplazar texto de una cadena.

Ejemplos

Primero, consideremos cómo eliminar etiquetas HTML `<p>` de una cadena:

```
import re

text = "<p> Esto es un párrafo </p>"
result = re.sub(r'<.*?p>', "", text)

print (result)
```

Aviso, reemplazamos las etiquetas con una cadena vacía ''. Este es un método común para eliminar texto. A continuación, eliminaremos el espacio en blanco del principio del texto. El espacio en blanco consta de cuatro espacios.

```
import re

text = " Esto es un párrafo"
result = re.sub(r'\s{4}', "", text)

print(result)
```

1.3. Tokenización

Para muchas tareas de procesamiento del lenguaje natural, necesitamos acceder a cada palabra en una cadena. Para acceder a cada palabra, primero tenemos que dividir el texto en componentes más pequeños. El método para dividir el texto en componentes más pequeños se llama *tokenización* y los componentes individuales se llaman *tokens*.

Algunas operaciones comunes que requieren la tokenización incluyen:

- Encontrar cuántas palabras u oraciones aparecen en el texto
- Determinar cuántas veces existe una palabra o frase específica
- Tener en cuenta qué términos es probable que coexistan

Si bien los "tokens" suelen ser palabras o términos individuales, también pueden ser frases u otros fragmentos de texto de tamaño.

Para tokenizar palabras individuales, podemos usar la función **word_tokenize()** de NLTK. La función acepta una cadena y devuelve una lista de palabras:

```
from nltk.tokenize import word_tokenize

text = "Tokenize this text"
tokenized = word_tokenize(text)

print(tokenized)
# ["Tokenize", "this", "text"]
```

Para *tokenizar* a nivel de oración, podemos usar **sent_tokenize()** del mismo módulo.

```
from nltk.tokenize import sent_tokenize

text = "Tokenize this sentence. Also, tokenize this sentence."
tokenized = sent_tokenize(text)

print(tokenized)
# ['Tokenize this sentence.', 'Also, tokenize this sentence.']
```

1.4. Normalización

La tokenización y la eliminación de ruido son elementos básicos de casi todas las tareas de filtrado en el preprocesamiento de texto. Sin embargo, algunos datos pueden requerir un procesamiento adicional mediante la normalización de texto. La *normalización* de un texto es un término general para varias tareas de preprocesamiento de texto:

- mayúsculas o minúsculas
- eliminación de palabras vacías
- *Stemming* : eliminando sin rodeos prefijos y sufijos de una palabra
- *Lematización* : reemplazando un token de una sola palabra con su raíz

El más simple de estos enfoques es cambiar mayúsculas y minúsculas. Podemos usar los métodos de cadena incorporados de Python para hacer ello:

```
my_string = 'tHiS HaS a MiX oF cAsEs'

print(my_string.upper())
# 'THIS HAS A MIX OF CASES'

print(my_string.lower())
# 'this has a mix of cases'
```

1.4.1. Eliminación de palabras irrelevantes

Las palabras irrelevantes son palabras que eliminamos durante el preprocesamiento cuando no nos importa la estructura de la oración. Por lo general, son las palabras más comunes en un idioma y no brindan información sobre el tono de una declaración. Incluyen palabras como "a", "un" y "los".

NLTK proporciona una librería incorporada con estas palabras. Puedes importarla usando la siguiente declaración:

```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
```

Creamos un conjunto con las palabras para que podamos verificar si las palabras están en una lista a continuación. Ahora que tenemos las palabras guardadas en *stop_words*, podemos usar la tokenización y una lista de comprensión para eliminarlas de una oración:

```
nbc_statement = "NBC was founded in 1926 making it the oldest major broadcast network in the USA"

word_tokens = word_tokenize(nbc_statement)
# tokenize nbc_statement

statement_no_stop = [word for word in word_tokens if word not in stop_words]

print(statement_no_stop)
# ['NBC', 'founded', '1926', 'making', 'oldest', 'major', 'broadcast', 'network', 'USA']
```

En este código, primero tokenizamos nuestra cadena, *nbc_statement*, luego usamos una lista de comprensión para devolver una lista con todas las palabras vacías eliminadas.

1.4.2. Derivación ("stemming")

En el procesamiento del lenguaje natural, la *derivación* es la tarea de normalización de preprocesamiento del texto que se ocupa de eliminar los afijos de las palabras (prefijos y sufijos). Por ejemplo, la derivación convertiría la palabra "cocinar" en "cocina". Este es un método común utilizado por los motores de búsqueda para mejorar la coincidencia entre la entrada del usuario y las visitas al sitio web.

NLTK tiene un *derivador* ("stemmer") incorporado llamado **PorterStemmer**. Puede usarlo con una lista de comprensión para derivar cada palabra en una lista de palabras simbólicas.

Primero, debe importar e inicializar el lematizador:

```
from nltk.stem import SnowballStemmer

stemmer = SnowballStemmer('spanish')
```

```
words_stemmed = [ stemmer.stem(token) for token in words_cleaned ]

print(words_stemmed)
# ['esta', 'primer', 'prueb', 'realiz', 'elimin', 'palabr', 'irrelev', 'oracion']
```

El hecho de que estas palabras se hayan reducido es útil para muchas aplicaciones de procesamiento de lenguaje. Sin embargo, se debe tener cuidado al derivar cadenas, porque las palabras a menudo se pueden convertir en algo irreconocible.

1.4.3. Lematización

La *lematización* es un método de conversión de palabras a su forma raíz. Este es un proceso más complicado que la derivación, porque requiere que el método conozca la parte gramatical de cada palabra. Dado que la lematización requiere la parte del discurso, es un enfoque menos eficiente que la derivación.

Podemos usar **WordNetLemmatizer** de NLTK para lematizar texto:

```
from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()
```

Una vez que tenemos el *lematizador* inicializado, podemos usar una lista de comprensión para aplicar la operación lematizar a cada palabra en una lista:

```
tokenized = ["NBC", "was", "founded", "in", "1926"]

lemmatized = [lemmatizer.lemmatize(token) for token in tokenized]

print(lemmatized)

# ["NBC", "wa", "founded", "in", "1926"]
```

El resultado, guardado como lematizado, contiene 'wa', mientras que el resto de las palabras siguen siendo las mismas. No es muy útil. Esto sucedió porque **lemmatize()** trata cada palabra como un sustantivo. Para aprovechar el poder de la lematización, debemos etiquetar cada palabra de nuestro texto con la parte más probable del discurso.

Si queremos trabajar en español **WordNetLemmatizer** no da buenos resultados y deberemos usar otra librería. Optaremos por la librería de Stanford llamada "Stanza" para realizar la lematización.

```
import stanza

stanza.download("es")

nlp = stanza.Pipeline(lang='es', processors='tokenize,mwt,pos,lemma')

example = "Esta es la primera prueba que realizo eliminando palabras irrelevantes de una oración"

doc = nlp(example)

words_lemmatized = [ [word.text, word.lemma] for sent in doc.sentences for word in sent.words ]

for words in words_lemmatized:

    print("Palabra: ", "{:15}".format(words[0]).lower(), "Lema: ", "{:15}".format(words[1]).lower())
```

1.4.4. Etiquetado de localización en el discurso ("Part-of-Speech Tagging")

Para mejorar el rendimiento de la lematización, necesitamos localizar cada palabra en el lugar del discurso que ocupa en nuestra cadena de texto.

En el siguiente script creamos el código que realiza un etiquetado de parte del discurso.

```
from nltk.tokenize import word_tokenize

from nltk.stem import WordNetLemmatizer

lemmatizer = WordNetLemmatizer()

populated_island = 'Indonesia was founded in 1945. It contains the most populated island in the world, Java, with over 140 million people.'

tokenized_string = word_tokenize(populated_island)

lemmatized_pos = [ lemmatizer.lemmatize(token, get_part_of_speech(token)) for token in tokenized_string ]

try:

    print(f'The lemmatized words are: {lemmatized_pos}')

except:
```

```
print('Expected a variable called `lemmatized_pos`')
```

Este script se apoya en un script ("get_part_of_speech") que realiza la parte de etiquetado como tal. El siguiente código visualiza esta función:

```
from collections import Counter
from nltk.corpus import wordnet

def get_part_of_speech(item):
    probable_part_of_speech = wordnet.synsets(item)
    pos_counts = Counter()
    pos_counts["n"] = len ([ item for item in probable_part_of_speech if item.pos()
    == "n" ])
    pos_counts["v"] = len ([ item for item in probable_part_of_speech if item.pos()
    == "v" ])
    pos_counts["a"] = len ([ item for item in probable_part_of_speech if item.pos()
    == "a" ])
    pos_counts["r"] = len ([ item for item in probable_part_of_speech if item.pos()
    == "r" ])
    most_probable_part_of_speech = pos_counts.most_common(1) [0] [0]
    return most_probable_part_of_speech
```

La función acepta una palabra y luego devuelve la parte más común de la oración para esa palabra.

Veámoslo por pasos:

1.4.4.1. Importar wordnet y Counter

```
from nltk.corpus import wordnet
from collections import Counter
```

- *wordnet* es una base de datos que usamos para contextualizar palabras
- *Counter* es un contenedor que almacena elementos como claves de diccionario

1.4.4.2. Obtener sinónimos

Dentro de nuestra función, usamos la función *wordnet.synsets()*¹ para obtener un conjunto de sinónimos para la palabra que se le pasa como argumento:

```
def get_part_of_speech(word):
```

¹ Para explicación con ejemplos de "Synsets":

<https://www.geeksforgeeks.org/nlp-synsets-for-a-word-in-wordnet/>


```
probable_part_of_speech = wordnet.synsets(word)
```

Los sinónimos devueltos vienen con su parte gramatical.

1.4.4.3. Utiliza sinónimos para determinar la parte más probable del discurso.

A continuación, creamos un objeto *Counter()* y establecemos cada valor en el recuento de la cantidad de sinónimos que se encuentran en cada parte del discurso. Para ello se tiene en cuenta cómo 'synet' organiza la información.

Cada tipo de término se corresponde con estas equivalencias:

Part of Speech	Tag
Noun	n
Verb	v
Adjective	a
Adverb	r

Si queremos contar el número de términos, por ejemplo, que son sustantivos, podríamos hacerlo de la siguiente forma:

```
pos_counts["n"] = len( [ item for item in probable_part_of_speech if item.pos()=="n" ] )
...
```

1.4.4.4. Devolver la parte más común de la oración

Ahora que tenemos un recuento para cada parte de la oración, podemos usar el método de contador *.most_common()* para encontrar y devolver la función gramatical más común en el discurso donde se ubica:

```
most_likely_part_of_speech = pos_counts.most_common(1)[0][0]
```

Averiguado lo anterior, podemos pasarlo a nuestro lematizador cuando encontremos la raíz de cada palabra. Echemos un vistazo a cómo haríamos esto para una cadena tokenizada:

```
tokenized = ["How", "old", "is", "the", "country", "Indonesia"]

lemmatized = [lemmatizer.lemmatize(token, get_part_of_speech(token)) for token in tokenized]

print(lemmatized)
```

```
# ['How', 'old', 'be', 'the', 'country', 'Indonesia']  
# Previously: ['How', 'old', 'is', 'the', 'country', 'Indonesia']
```

Debido a que pasamos en la parte del discurso, "is" (en inglés) se obtuvo su raíz, "be". Esto significa que palabras como "was" y "were" se convertirán en "be".

Resumen

Esta lección no es una introducción exhaustiva a preprocesamiento de texto. Sin embargo, muestra algunos de los trucos más comunes para limpiar datos. Antes de preprocesar texto, lo más importante es tener una idea de cómo se desea formatear los datos y por qué se desea que se haga de esa manera. Una vez que se sepa lo que se quiere, se pueden usar estas herramientas para hacerlo.

Repasemos lo que cubrimos en esta lección:

- El preprocesamiento de texto trata de limpiar y preparar datos de texto para que estén listos para otras tareas de PNL.
- La eliminación de ruido es un paso de preprocesamiento de texto que se ocupa de eliminar el formato innecesario de nuestro texto.
- La *tokenización* es un paso de preprocesamiento de texto dedicado a dividir el texto en unidades más pequeñas (generalmente palabras o términos discretos).
- *Normalización* es el nombre que le damos a la mayoría de las otras tareas de preprocesamiento de texto, incluidas la lematización, la lematización, las mayúsculas y minúsculas y la eliminación de palabras vacías.
 - *Derivación* ("stemming") es la tarea de preprocesamiento de normalización que se centra en eliminar los afijos de palabras.
 - La *lematización* es la tarea de preprocesamiento de normalización que lleva más cuidadosamente las palabras a su forma raíz.

Videos externos el preprocesamiento de texto basado en NLTK

Recomendamos ver los siguientes videos para obtener tutoriales útiles sobre el uso de NLTK para el preprocesamiento de texto:

- [Natural Language Processing With Python and NLTK p.1 Tokenizing words and Sentences](#)

En este video, obtendrá un tutorial sobre cómo tokenizar texto usando NLTK. Esto es útil si desea

ver una demostración de cómo dividir los datos de texto en palabras u oraciones.

- [Stop Words - Natural Language Processing With Python and NLTK p.2](#)

vacías: En este video, obtendrá un tutorial sobre cómo eliminar palabras vacías de datos de texto usando NLTK, que es útil si está preparando texto para análisis de opiniones, modelado de temas u otras tareas donde las palabras comunes no son útiles.

- [Stemming - Natural Language Processing With Python and NLTK p.3](#)
- [Lemmatizing - Natural Language Processing With Python and NLTK p.8](#)

En estos videos, obtendrá tutoriales sobre Stemming y lemmatization usando NLTK, que explican con más detalle las diferencias entre estas dos técnicas.