RAIn 2025
Trabajo Práctico Nº 2

Tema: Introducción - 2da parte

Fecha Inicio: 01/04/2025 **Fecha de Entrega:** 15/04/2025

Autor: Jorge Justino Riera **LU:** 5575 **Carrera:** Ing. Inf **Plan:** 2010

Repositorio código fuente: https://github.com/nicrom8b/rain/tree/main/tp2

Índice

1. Eliminar stop_words y tokenizacion

Empleando la librería NLTK de Python, elimine las stop_words empleando el idioma español, tokenize el texto anterior, y muestre el resultado con la frecuencia de cada término, ordenado por frecuencia descendente (además del listado, muestre un gráfico con los 20 términos/tokens más frecuentes)

Ejecución

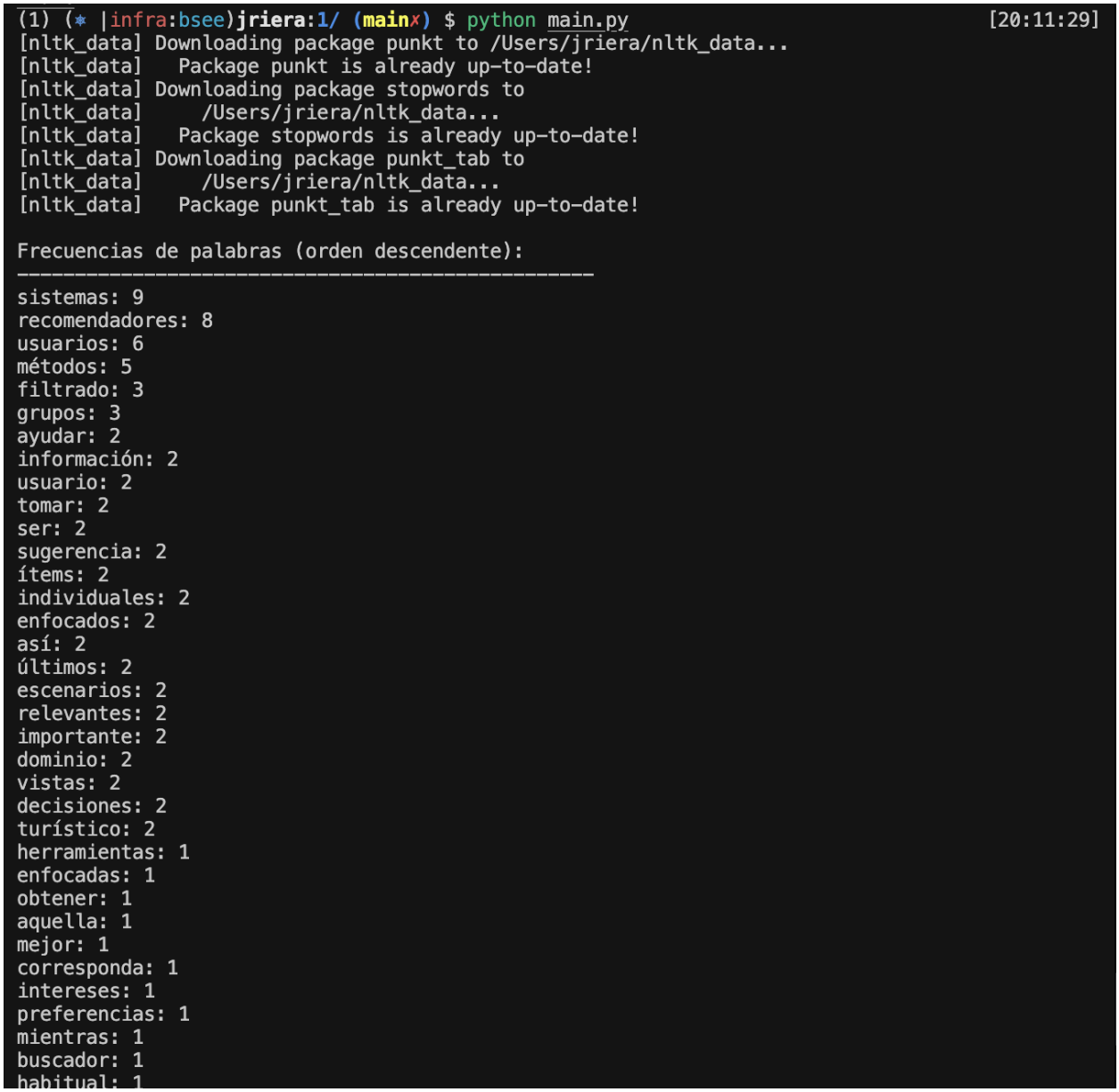
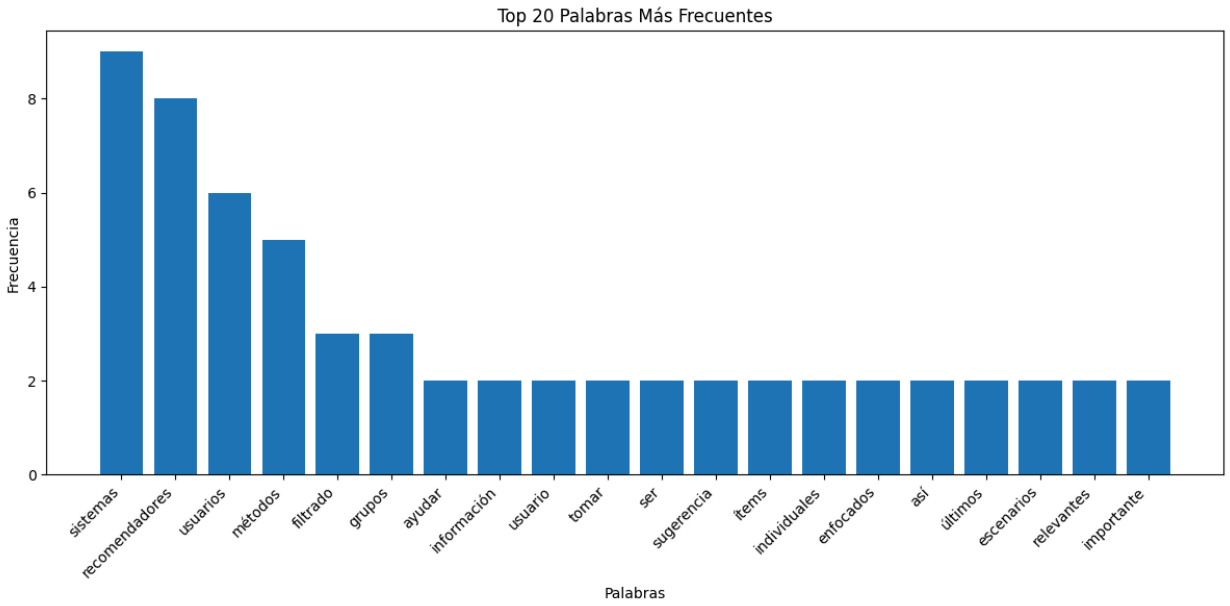


Gráfico 20 tokens mas frecuentes



Explicación

El proyecto cuenta con tres partes principales

- **Inicialización**
  En esta sección importamos bibliotecas necesarias (nltk, matplotlib, etc.) y se descargan los recursos de NLTK (punkt, stopwords y punkt_tab)
- **Procesamiento de texto**
  Esta es la parte central del script en la cual:
  - Se lee y procesa el archivo de texto
  - Se limpia el texto (minúsculas, sin puntuación)

    ```
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    ```
  - Se tokeniza y filtran las palabras utilizando las funciones `word_tokenize` y `stopwords` configuradas en Español.

    ```
    tokens = word_tokenize(text, language='spanish')
    stop_words = set(stopwords.words('spanish'))
    filtered_tokens = [word for word in tokens if word not in stop_words and word.strip()]
    ```
  - Se cuenta frecuencia de cada palabra

    ```
    word_freq = Counter(filtered_tokens)
    ```
- **Análisis y visualización**
  Por último esta sección trabajamos
  - Se imprimen los resultados

- Se genera un gráfico de barras
- Se guarda el gráfico como imagen, para ello utilizamos `matplotlib.pyplot` y se guarda en un archivo top_words.png

2. Stemming en Inglés

Del texto a continuación, aplique el proceso de eliminación de stop_words en inglés y tokenización, a continuación emplee el proceso de Stemming con los algoritmos de Porter y Lancaster, comparando los resultados de los dos procesos encolumnados (Texto 2).

Ejecución

```
(2) (* |infra:bsee)jriera:2/ (mainx) $ python main.py                    [20:58:28]
[nltk_data] Downloading package punkt to /Users/jriera/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/jriera/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

Comparación de Stemming (Porter vs Lancaster):
--------------------------------------------------------------------------------
+---------------+---------------+-------------------+
| Original      | Porter Stem   | Lancaster Stem    |
+===============+===============+===================+
| information   | inform        | inform            |
+---------------+---------------+-------------------+
| retrieval     | retriev       | retriev           |
+---------------+---------------+-------------------+
| process       | process       | process           |
+---------------+---------------+-------------------+
| obtaining     | obtain        | obtain            |
+---------------+---------------+-------------------+
| relevant      | relev         | relev             |
+---------------+---------------+-------------------+
| information   | inform        | inform            |
+---------------+---------------+-------------------+
| collection    | collect       | collect           |
+---------------+---------------+-------------------+
| data          | data          | dat               |
+---------------+---------------+-------------------+
| involves      | involv        | involv            |
+---------------+---------------+-------------------+
| searching     | search        | search            |
+---------------+---------------+-------------------+
| retrieving    | retriev       | retriev           |
+---------------+---------------+-------------------+
| information   | inform        | inform            |
+---------------+---------------+-------------------+
| various       | variou        | vary              |
+---------------+---------------+-------------------+
| sources       | sourc         | sourc             |
+---------------+---------------+-------------------+
```

Explicación

De la ejecución anterior además podemos ver algunas estadísticas que obtenemos al final **Estadísticas**:

Número total de tokens: 101
Número de tokens únicos: 82
Número de stems únicos (Porter): 78
Número de stems únicos (Lancaster): 78

Y centrándonos en la comparativa de alguno de los tokes que presente variaciones por ejemplo: | various | variou | vary |

- **Original (various):** Esta es la palabra original en su forma completa (extraída del texto).
- **Porter Stem (variou)**: El algoritmo de Porter es más conservador y suave en su enfoque. En este caso:
    - Elimina la 's' final
    - Mantiene la raíz "variou"
    - Este algoritmo tiende a ser más conservador para preservar la semántica de la palabra
- **Lancaster Stem (vary):** El algoritmo de Lancaster es más agresivo y radical:
    - Reduce la palabra a su forma más básica "vary"
    - Es más agresivo en la eliminación de sufijos
    - Tiende a producir stems más cortos y radicales

La diferencia principal entre estos resultados se debe a las diferentes reglas y enfoques de cada algoritmo:
Porter es más conservador y mantiene más de la forma original de la palabra Lancaster es más agresivo y busca reducir la palabra a su forma más básica posible.

Función central

La función principal para este proyecto es `apply_stemming`:

```
def apply_stemming(tokens):
    """Aplica los algoritmos de stemming Porter y Lancaster a los tokens"""
    porter = PorterStemmer()
    lancaster = LancasterStemmer()

    porter_stems = [porter.stem(token) for token in tokens]
    lancaster_stems = [lancaster.stem(token) for token in tokens]

    return porter_stems, lancaster_stems
```

Utilizando `**nltk.stem**` inicializamos los procesos de porter y lancaster, para luego procesar los tokens que se reciben por parámetro.
La funcion devuelve dos listas `**porter_stems**` y `**lancaster_stems**` de los stems generados.

3. Stemming en Español

Aplique el proceso de Stemming para el Texto 1 y muestre el resultado. Advierta si los algoritmos de Porter y Lancaster en NLTK poseen la implementación para el idioma español, sino es así, aplique otro algoritmo que si la posea.

Ejecución

```
(3) (* |infra:bsee)jriera:3/ (mainx) $ python main.py
[21:31:13]
[nltk_data] Downloading package punkt to /Users/jriera/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/jriera/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

Comparación de diferentes algoritmos de stemming para español:
```

Original	Snowball	Porter	SpaCy
sistemas	sistem	sistema	sistema
recomendadores	recomend	recomendador	recomendador
herramientas	herramient	herramienta	herramienta
enfocadas	enfoc	enfocada	enfocado
ayudar	ayud	ayudar	ayudar
usuarios	usuari	usuario	usuario
obtener	obten	obten	obtener
aquella	aquell	aquella	aquel
información	inform	información	información
mejor	mejor	mejor	mejor
corresponda	correspond	corresponda	correspondo
intereses	interes	interes	interés
preferencias	preferent	preferencia	preferencia
mientras	mientr	mientra	mientras
buscador	buscador	buscador	buscador
habitual	habitual	habitu	habitual
centra	centr	centra	centrar

Explicación

Este proyecto procesa un texto en español utilizando diferentes algoritmos de stemming para:
1. Eliminar palabras vacías (stopwords) en español.

2. Tokenizar el texto.
3. Aplicar diferentes algoritmos de stemming/lemmatización.
4. Comparar los resultados de cada algoritmo.

Los algoritmos de Porter y Lancaster en NLTK no tienen implementación para el idioma español. Estos algoritmos están diseñados específicamente para el idioma inglés y no son adecuados para otros idiomas debido a las diferencias en la morfología de las palabras.

En NLTK, el único algoritmo de stemming disponible para español es Snowball (también conocido como Porter2), que:
- Tiene soporte específico para español
- Es considerado una mejora del algoritmo Porter original
- Está diseñado para manejar las particularidades morfológicas del español
- Es el stemmer recomendado por NLTK para el procesamiento de texto en español

Algoritmos Implementados en la funcion central apply_stemming

1. **Snowball (Porter2) Stemmer (NLTK)**
    a. Implementación de NLTK
    b. Especializado para español
    c. Es considerado una mejora del algoritmo Porter original
    d. Reduce las palabras a su raíz eliminando sufijos y prefijos
    Ejemplo: "corriendo" → "corr", "caminando" → "camin"

```
snowball = SnowballStemmer('spanish')
snowball_stems = [snowball.stem(token) for token in tokens]
```

2. **Porter Stemmer (stemming)**
    a. Implementación del algoritmo Porter original
    b. Aunque está diseñado para inglés, se lo incluye para comparación
    c. Más agresivo que Snowball
    d. Reduce palabras a su forma base
    Ejemplo: "corriendo" → "corr", "caminando" → "camin"

```
porter_stems = [stem(token) for token in tokens]
```

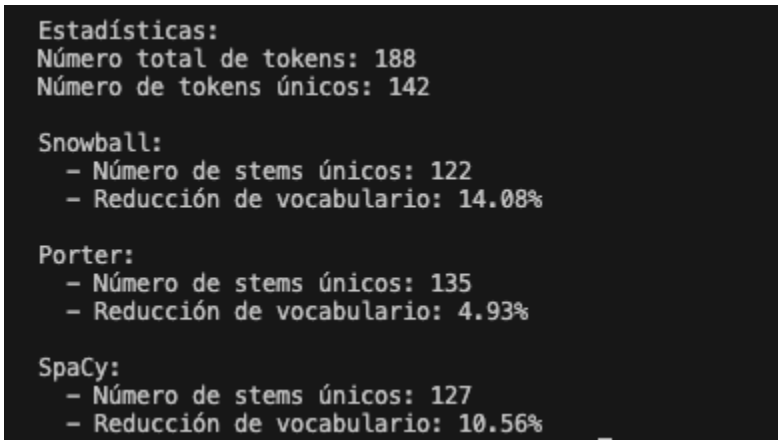
3. **SpaCy Lemmatization**
    a. Usa modelos de lenguaje entrenados específicamente para español
    b. Proporciona lemas en lugar de stems
    c. Considerado más preciso pero más lento
    d. Convierte las palabras a su forma base según el diccionario
    Ejemplo: "corriendo" → "correr", "caminando" → "caminar"

```
nlp = spacy.load('es_core_news_sm')
doc = nlp(' '.join(tokens))
spacy_lemmas = [token.lemma_ for token in doc]
```

La función devuelve un diccionario con tres claves, cada clave contiene la lista de palabras reducidas por su respectivo algoritmo.

Estadísticas

De la comparativa de los algoritmos obtenemos las siguientes estadísticas:



4. N-gramas

Del primer párrafo del Texto 1, obtenga 2-gramas y 3-gramas de palabras, muestre los resultados en cada caso.

Ejecución

El proyecto implementa un analizador de n-gramas para textos en español. Primeramente se realiza la extracción del primer párrafo del texto 1. Para luego generar dos tablas en laque se muestran bigramas y trigramas con sus frecuencias.

```
(4) (* |infra:bsee)jriera:4/ (mainx) $ python main.py                                                    [22:12:41]
[nltk_data] Downloading package punkt to /Users/jriera/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/jriera/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

Primer párrafo del texto:
--------------------------------------------------------------------------------
Los sistemas recomendadores son herramientas enfocadas a ayudar a los usuarios a obtener aquella información que mejor
 se corresponda con sus intereses y preferencias. Mientras que un  buscador  habitual  se  centra  en  encontrar  aque
llo  que  el  usuario  solicita,  un  sistema recomendador  ayuda  al  usuario  a  tomar  una  decisión,  que  puede
ser  la  compra  de  un producto  en  un  portal  de  comercio  electrónico,  la  lectura  de  un  libro,  la  revisió
n  de  un artículo  científico,  el  acceso  a  una  página  web  en  específico,  o  el  estudio  de  determinado rec
urso educativo en una plataforma virtual de aprendizaje.
--------------------------------------------------------------------------------

2-gramas (bigramas) más frecuentes:
+----------------------------+--------------+
| N-gram                     |   Frecuencia |
+============================+==============+
| sistemas recomendadores    |            1 |
+----------------------------+--------------+
| tomar decisión             |            1 |
+----------------------------+--------------+
| puede ser                  |            1 |
+----------------------------+--------------+
| ser compra                 |            1 |
+----------------------------+--------------+
| compra producto            |            1 |
+----------------------------+--------------+
| producto portal            |            1 |
+----------------------------+--------------+
| portal comercio            |            1 |
+----------------------------+--------------+
| comercio electrónico       |            1 |
+----------------------------+--------------+
| electrónico lectura        |            1 |
+----------------------------+--------------+
| lectura libro              |            1 |
+----------------------------+--------------+
| libro revisión             |            1 |
+----------------------------+--------------+
| revisión artículo          |            1 |
+----------------------------+--------------+
```

3-gramas (trigramas) más frecuentes:

N-gram	Frecuencia
sistemas recomendadores herramientas	1
recomendadores herramientas enfocadas	1
decisión puede ser	1
puede ser compra	1
ser compra producto	1
compra producto portal	1
producto portal comercio	1
portal comercio electrónico	1
comercio electrónico lectura	1
electrónico lectura libro	1
lectura libro revisión	1
libro revisión artículo	1
revisión artículo científico	1
artículo científico acceso	1
científico acceso página	1
acceso página web	1
página web específico	1
web específico estudio	1
específico estudio determinado	1

Explicación

La función principal de este proyecto es **generate_ngrams(tokens, n)** y es relativamente simple pero la más relevante para esta consigna.
Recibe como parametros:

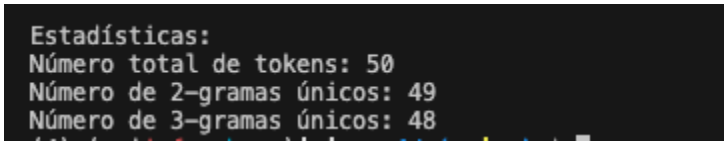
- **tokens**: Una lista de palabras/tokens ya procesados (sin stopwords, puntuación, etc.)
- **n**: Un número entero que indica el tamaño del n-grama que queremos generar

Funcionamiento:

Utiliza la función **nltk.ngrams()** de la biblioteca NLTK. Esta función toma la lista de tokens y genera todas las secuencias posibles de n palabras consecutivas. El resultado se convierte a lista con **list()**

```
def generate_ngrams(tokens, n):
    """Generar n-gramas a partir de tokens"""
    return list(nltk.ngrams(tokens, n))
```

De la ejecución anterior obtenemos ciertas estadísticas:



5. Detokenización

Empleando el corpus Brown de NLTK, detokenize el archivo cg73.

a. Tokenize en oraciones.
b. Muestre las primeras 10.

Ejecución

```
(* |infra:bsee)jriera:5/ (mainx) $  . /Users/jriera/.local/share/virtualenvs/5-eBDrNYlE/bin/activate       [22:23:05]
(5) (* |infra:bsee)jriera:5/ (mainx) $ python main.py                                                    [22:23:05]
[nltk_data] Downloading package brown to /Users/jriera/nltk_data...
[nltk_data]   Package brown is already up-to-date!
[nltk_data] Downloading package punkt to /Users/jriera/nltk_data...
[nltk_data]   Package punkt is already up-to-date!

Primeras 10 oraciones del archivo cg73:
--------------------------------------------------------------------------------

Oración 1:
The recent experiments in the new poetry-and-jazz movement seen by some as part of the " San Francisco Renaissance " have been as popular
as they are notorious.
"
--------------------------------------------------------------------------------

Oración 2:
It might well start a craze like swallowing goldfish or pee-wee golf ", wrote Kenneth Rexroth in an explanatory note in the Evergreen Revi
ew, and he may have been right.
--------------------------------------------------------------------------------

Oración 3:
Under the general heading " poetry-and-jazz " widely divergent experiments have been carried out.
--------------------------------------------------------------------------------

Oración 4:
Lawrence Ferlenghetti and Bruce Lippincott have concentrated on writing a new poetry for reading with jazz that is very closely related to
 both the musical forms of jazz, and the vocabulary of the musician.
--------------------------------------------------------------------------------

Oración 5:
Even musicians themselves have taken to writing poetry.
--------------------------------------------------------------------------------

Oración 6:
(( Judy Tristano now has poems as well as ballads written for her.
))
--------------------------------------------------------------------------------

Oración 7:
But the best known exploiters of the new medium are Kenneth Rexroth and Kenneth Patchen.
--------------------------------------------------------------------------------

Oración 8:
Rexroth and Patchen are far apart musically and poetically in their$ experiments.
--------------------------------------------------------------------------------

Oración 9:
Rexroth is a longtime jazz buff, a name-dropper of jazz heroes, and a student of traditional as well as modern jazz.
--------------------------------------------------------------------------------

Oración 10:
In San Francisco he has worked with Brew Moore, Charlie Mingus, and other " swinging " musicians of secure reputation, thus placing himsel
f within established jazz traditions, in addition to being a part of the San Francisco " School ".
--------------------------------------------------------------------------------

Todas las oraciones han sido guardadas en output.txt
```

Explicación

El script realiza las siguientes tareas:

1. Descarga los recursos necesarios de NLTK (corpus Brown y tokenizador de oraciones).
2. Obtiene el texto del archivo cg73 del corpus Brown.
3. Realiza una limpieza/eliminación de etiquetas POS(Part-of-Speech tags) y caracteres especiales.
4. Tokeniza el texto en oraciones.
5. Muestra las primeras 10 oraciones.
6. Muestra estadísticas sobre el número total de oraciones.
7. Guarda las oraciones en un archivo de texto, una por línea.

La función principal es **sent_tokenize(cleaned_text).** Es una función especializada en la segmentación de texto en oraciones. Es parte del módulo **punkt de NLTK**, que es un algoritmo entrenado para reconocer los límites de las oraciones en diferentes idiomas.
En este caso específico: **cleaned_text** es el texto que ya ha sido limpiado previamente (eliminando etiquetas POS, caracteres especiales, etc.). El resultado se almacena en la variable **sentences**, que será una lista de strings, donde cada string es una oración completa.

6. Proceso de comparación

Realice paso a paso el preprocesamiento del texto obtenido en el punto anterior, ello incluye:

a. Eliminación de ruido
b. Tokenización
c. Normalización
d. Eliminación de palabras vacías
e. Obtener un listado de las 50 palabras más frecuentes
f. Stemming. Obtener un listado de las 50 palabras más frecuentes
g. Lematización. Obtener un listado de las 50 palabras más frecuentes
h. Lematización indicando el PoS para los verbos.
i. Realizar una representación tabular de los primeros 30 tokens indicando la palabra normal, realizado el stemming, lematización y lematización con PoS (verbos)

Ejecución


(6) (* |infra:bsee)jriera:6/ (mainx) $ python main.py [22:36:40]
Descargando recursos de NLTK...
[nltk_data] Downloading package punkt to /Users/jriera/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data] /Users/jriera/nltk_data...
[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/jriera/nltk_data...
[nltk_data] Package wordnet is already up-to-date!
Recursos descargados correctamente.
A. Texto después de eliminar ruido:
The recent experiments in the new poetry and jazz movement seen by some as part of the San Francisco Renaissance have been as popular as they are notor
ious It might well start a craze like swallowing ...
B. Primeros 10 tokens:
['the', 'recent', 'experiments', 'in', 'the', 'new', 'poetry', 'and', 'jazz', 'movement']
C. Primeros 10 tokens normalizados:
['the', 'recent', 'experiments', 'in', 'the', 'new', 'poetry', 'and', 'jazz', 'movement']
D. Primeros 10 tokens sin palabras vacías:
['recent', 'experiments', 'new', 'poetry', 'jazz', 'movement', 'seen', 'part', 'san', 'francisco']

```
E. 50 palabras más frecuentes:
--------------------------------------------------------------------------------
jazz: 35
patchen: 30
poetry: 21
poems: 12
musician: 11
new: 10
private: 10
experiments: 9
musicians: 8
many: 8
detective: 8
hero: 8
well: 7
like: 7
rexroth: 7
myth: 7
anger: 6
may: 5
first: 5
work: 5
almost: 5
love: 5
kind: 5
symbols: 5
must: 5
angry: 5
san: 4
francisco: 4
kenneth: 4
reading: 4
modern: 4
often: 4
early: 4
form: 4
approach: 4
book: 4
world: 4
man: 4
death: 4
art: 4
feeling: 4
one: 4
hold: 4
han: 4
seen: 3
writing: 3
tristano: 3
written: 3
school: 3
although: 3
```

```
F. 50 palabras más frecuentes después de stemming:
--------------------------------------------------------------------------------
jazz: 35
patchen: 30
poetri: 21
musician: 19
poem: 15
experi: 11
new: 10
privat: 10
hero: 9
work: 9
detect: 9
read: 8
myth: 8
mani: 8
well: 7
like: 7
rexroth: 7
music: 6
form: 6
love: 6
use: 6
anger: 6
may: 5
write: 5
first: 5
almost: 5
kind: 5
eye: 5
symbol: 5
draw: 5
must: 5
angri: 5
san: 4
francisco: 4
kenneth: 4
poetic: 4
modern: 4
school: 4
often: 4
earli: 4
approach: 4
book: 4
public: 4
world: 4
man: 4
death: 4
make: 4
refer: 4
art: 4
feel: 4
```

```
G. 50 palabras más frecuentes después de lematización:
--------------------------------------------------------------------------------
jazz: 35
patchen: 30
poetry: 21
musician: 19
poem: 15
new: 10
private: 10
experiment: 9
hero: 9
detective: 9
myth: 8
many: 8
well: 7
like: 7
rexroth: 7
work: 7
form: 6
love: 6
anger: 6
may: 5
first: 5
almost: 5
kind: 5
eye: 5
symbol: 5
must: 5
angry: 5
san: 4
francisco: 4
kenneth: 4
reading: 4
modern: 4
school: 4
often: 4
read: 4
early: 4
approach: 4
book: 4
world: 4
man: 4
death: 4
reference: 4
art: 4
feeling: 4
one: 4
hold: 4
han: 4
seen: 3
writing: 3
tristano: 3
```

```
H. 50 palabras más frecuentes después de lematización con PoS:
--------------------------------------------------------------------------------
jazz: 35
patchen: 30
poetry: 21
musician: 19
poem: 15
experiment: 11
new: 10
private: 10
write: 9
hero: 9
work: 9
detective: 9
read: 8
myth: 8
many: 8
well: 7
like: 7
rexroth: 7
form: 6
know: 6
love: 6
anger: 6
may: 5
take: 5
first: 5
almost: 5
kind: 5
eye: 5
symbol: 5
use: 5
must: 5
angry: 5
san: 4
francisco: 4
kenneth: 4
modern: 4
school: 4
often: 4
early: 4
begin: 4
approach: 4
book: 4
world: 4
man: 4
death: 4
make: 4
reference: 4
art: 4
draw: 4
feel: 4
```

I. Tabla comparativa de los primeros 30 tokens:

Palabra Original	Stemming	Lematización	Lematización con PoS
recent	recent	recent	recent
experiments	experi	experiment	experiment
new	new	new	new
poetry	poetri	poetry	poetry
jazz	jazz	jazz	jazz
movement	movement	movement	movement
seen	seen	seen	see
part	part	part	part
san	san	san	san
francisco	francisco	francisco	francisco
renaissance	renaiss	renaissance	renaissance
popular	popular	popular	popular
notorious	notori	notorious	notorious
might	might	might	might
well	well	well	well
start	start	start	start
craze	craze	craze	craze
like	like	like	like
swallowing	swallow	swallowing	swallow
goldfish	goldfish	goldfish	goldfish
pee	pee	pee	pee
wee	wee	wee	wee
golf	golf	golf	golf
wrote	wrote	wrote	write
kenneth	kenneth	kenneth	kenneth
rexroth	rexroth	rexroth	rexroth
explanatory	explanatori	explanatory	explanatory
note	note	note	note
evergreen	evergreen	evergreen	evergreen
review	review	review	review

(6) (* |infra:bsee)jriera:6/ (mainx) $

Explicación

El script va generando paso a paso lo solicitado en la consigna. Este es un programa de procesamiento de texto (NLP - Natural Language Processing) que utiliza la biblioteca **NLTK** (Natural Language Toolkit) para analizar y procesar texto. El programa realiza una serie de transformaciones y análisis sobre un texto de entrada, específicamente leyendo de un archivo llamado output.txt.
Las principales etapas del procesamiento son:

1. Preparación inicial:
    a. Descarga los recursos necesarios de NLTK (tokenizador, stopwords y wordnet)
    b. Lee el texto de entrada desde un archivo
2. Limpieza del texto:
    a. Elimina caracteres especiales y números
    b. Normaliza espacios en blanco
3. Tokenización y normalización
    a. Divide el texto en palabras individuales (tokens)
    b. Convierte todo a minúsculas
    c. Filtra para mantener solo palabras alfabéticas
    d.
4. Filtrado de palabras
    a. Elimina palabras comunes (stopwords) que no aportan significado
5. Análisis de frecuencia
    a. Cuenta y muestra las 50 palabras más frecuentes
6. Procesamiento lingüístico
    a. Aplica stemming (reducción de palabras a su raíz)
    b. Aplica lematización (conversión a la forma base de la palabra)
    c. Aplica lematización avanzada considerando la categoría gramatical
7. Comparación de resultados:
    a. Genera una tabla comparativa mostrando cómo las palabras se transforman en cada etapa del proceso