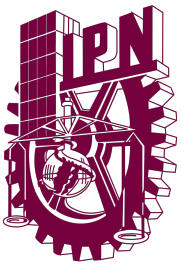

Instituto Politécnico Nacional

Escuela Superior de Cómputo

Práctica 1

R E P O R T E

Materia:

Tecnologías de Lenguaje Natural

presenta:

Luis Fernando Rodríguez-Domínguez

Profesor:

Ituriel Enrique Flores Estrada

Objetivo:

Documentar y evidenciar el desarrollo y la ejecución de programas orientados al procesamiento de textos. Se exploran técnicas a partir de expresiones regulares, y pipelines de normalización de textos

Índice general

Capítulo 1

Introducción

El presente reporte documenta la realización de la Práctica 1 de la asignatura de Tecnologías de Lenguaje Natural, en la cual se abordan dos tareas fundamentales en el procesamiento de datos textuales: el uso de expresiones regulares para la extracción de patrones específicos y la normalización de textos en inglés y español.

En la primera parte de la práctica se desarrolló un conjunto de scripts en Python para aplicar diversas expresiones regulares sobre un archivo de datos (Anexo "A"). Dichas expresiones permitieron identificar patrones concretos, como la presencia de secuencias específicas de caracteres, validación de formatos numéricos en descripciones de gastos, y transformaciones de direcciones IPv6, entre otros. El objetivo principal fue demostrar la capacidad de las expresiones regulares para filtrar y extraer información relevante de un conjunto de registros.

La segunda parte se centra en la normalización de textos, utilizando tres módulos destacados en el procesamiento de lenguaje natural: NLTK, spaCy y Stanza. Se realizó una investigación comparativa que incluyó el análisis exploratorio de textos originales y normalizados, la aplicación de técnicas de tokenización, remoción de stop words, stemming, lematización y otros filtros adicionales. La elección y el orden de aplicación de estas técnicas se justifican en función de su impacto en la homogeneización del contenido, lo que resulta crucial para el análisis posterior de corpus en distintos idiomas.

Este reporte presenta, en primer lugar, los resultados obtenidos mediante la aplicación de expresiones regulares, y posteriormente, se expone el proceso de normalización con un enfoque comparativo entre los distintos módulos, apoyado en gráficos y estadísticas que evidencian el rendimiento y la eficacia de cada técnica.

Capítulo 2

Desarrollo

2.1. Parte 1: Expresiones Regulares

2.1.1. Objetivo

El objetivo de esta sección es aplicar un conjunto de expresiones regulares sobre el archivo de datos (Anexo A) para extraer información específica de acuerdo con distintos criterios. Se implementaron nueve ejercicios que abarcan desde la búsqueda de patrones simples hasta la manipulación de formatos complejos como direcciones IPv6 y la inserción de separadores en números grandes.

2.1.2. Metodología

Se desarrollaron scripts en Python utilizando la librería `re` para definir y aplicar patrones sobre las líneas de datos del archivo. Cada ejercicio se abordó de la siguiente manera:

- **Lectura de datos:** Se lee el archivo `AnexoA.txt` y se separa el encabezado de los registros. Esto para los ejercicios del 1 al 5.
- **Definición de patrones:** Para cada ejercicio se define un patrón de expresión regular que cumpla con la condición requerida.
- **Filtrado de registros:** Se aplican los patrones sobre cada línea y se recogen aquellas que satisfacen las condiciones.
- **Visualización de resultados:** Se muestran los resultados (salidas) en pantalla mediante capturas, evidenciando la correcta ejecución del código.

2.1.3. Resultados y Evidencias

A continuación, se describe brevemente cada ejercicio implementado:

Ejercicio 1: Líneas con 'r' seguida de 'g'

1. Contengan una "r" seguida por una "g" (minúsculas). La "r" y la "g" no necesariamente tienen que estar en posiciones consecutivas. **(2 puntos)**.

```python
patron1 = re.compile(r"r.*g")  # Solo busca letras minúsculas

resultado1 = [linea for linea in datos if patron1.search(linea)]
print("\nPunto 1: Líneas con 'r' seguida de 'g':")
for r in resultado1:
    print(r)
```

✓ 0.0s Python

```
Punto 1: Líneas con 'r' seguida de 'g':
4:383.75:travel:20170223:flight to Boston, to visit ABC Corp.
5:55.00:travel:20170223:cab to ABC Corp. in Cambridge, MA
6:23.25:meal:20170223:dinner at Logan Airport
14:6.53:meal:20170302:Dunkin Donuts, drive to Big Inc. near DC
17:86.00:travel:20170304:mileage, drive to/from Big Inc., Reston, VA
19:378.81:travel:20170304:Hyatt Hotel, Reston VA, for Big Inc. meeting
25:86.00:travel:20170317:mileage, drive to/from Big Inc., Reston, VA
26:32.27:meal:20170317:lunch at Clyde's with Fred and Gina, Big Inc.
```

Figura 2.1: Resultado del ejercicio 1

Ejercicio 2: Comidas con costo mayor o igual a 100.00

2. Describan comidas que cuesten al menos 100.00. **(2 puntos)**.

```python
patron2 = re.compile(r'\:[1-9][0-9]{2}\.\d*\:meal')

resultado2 = [linea for linea in datos if patron2.search(linea)]

print("\nPunto 2: Comidas con costo >= 100.00:")
for r in resultado2:
    print(r)
```

✓ 0.0s Pyth

```
Punto 2: Comidas con costo >= 100.00:
8:142.12:meal:20170226:host dinner with ABC clients, Al, Bob, Cy, Dave, Ellie
15:127.23:meal:20170302:dinner, Tavern64
```

Figura 2.2: Resultado del ejercicio 2

Ejercicio 3: Patrón 'a' seguida de 'b' seguida de 'c'

```python
patron3 = re.compile(r'[aA][^aAbB]*[bB][^bBcC]*c')

resultado3 = [linea for linea in datos if patron3.search(linea)]
print("\nPunto 3: Líneas que contengan 'a', luego 'b' y luego 'c' con las restricciones:")
for r in resultado3:
    print(r)
```

[4] ✓ 0.0s Python

```
Punto 3: Líneas que contengan 'a', luego 'b' y luego 'c' con las restricciones:
14:6.53:meal:20170302:Dunkin Donuts, drive to Big Inc. near DC
17:86.00:travel:20170304:mileage, drive to/from Big Inc., Reston, VA
19:378.81:travel:20170304:Hyatt Hotel, Reston VA, for Big Inc. meeting
25:86.00:travel:20170317:mileage, drive to/from Big Inc., Reston, VA
26:32.27:meal:20170317:lunch at Clyde's with Fred and Gina, Big Inc.
```

Figura 2.3: Resultado del ejercicio 3

Ejercicio 4: Descripciones con 'a' y un dígito

```python
patron4 = re.compile(r'\:[^:]*a[^:]*\d[^:]*$|\:[^:]*\d[^:]*a[^:]*$')
resultado4 = [linea for linea in datos if patron4.search(linea)]

print("\nPunto 4: Líneas cuya descripción contenga una 'a' minúscula y un dígito:")
for r in resultado4:
    print(r)
```

✓ 0.0s Python

```
Punto 4: Líneas cuya descripción contenga una 'a' minúscula y un dígito:
12:79.99:supply:20170227:spare 20" monitor
15:127.23:meal:20170302:dinner, Tavern64
20:1247.49:supply:20170306:Dell 7000 laptop/workstation
24:195.89:supply:20170309:black toner, HP 304A, 2-pack
```

Figura 2.4: Resultado del ejercicio 4

Ejercicio 5: Patrón 'd' seguido de 'i'

```python
patron5 = re.compile(r'[dD].*[iI]')

resultado5 = [linea for linea in datos if patron5.search(linea)]
print("\nPunto 5: Líneas que contengan 'd' seguido de cualquier carácter y luego 'i':")
for r in resultado5:
    print(r)
```

✓ 0.0s Python

```
Punto 5: Líneas que contengan 'd' seguido de cualquier carácter y luego 'i':
6:23.25:meal:20170223:dinner at Logan Airport
8:142.12:meal:20170226:host dinner with ABC clients, Al, Bob, Cy, Dave, Ellie
14:6.53:meal:20170302:Dunkin Donuts, drive to Big Inc. near DC
15:127.23:meal:20170302:dinner, Tavern64
16:33.07:meal:20170303:dinner, Uncle Julio's
17:86.00:travel:20170304:mileage, drive to/from Big Inc., Reston, VA
20:1247.49:supply:20170306:Dell 7000 laptop/workstation
21:6.99:supply:20170306:HDMI cable
22:212.06:util:20170308:Duquesne Light
23:23.86:supply:20170309:Practical Guide to Quant Finance Interviews
25:86.00:travel:20170317:mileage, drive to/from Big Inc., Reston, VA
26:32.27:meal:20170317:lunch at Clyde's with Fred and Gina, Big Inc.
```

Figura 2.5: Enter Caption

Ejercicio 6: Títulos de películas producidas antes de 2002

```python
patron_pelicula = re.compile(r"\((19\d{2}|200[01])\)")

peliculas_antes_2002 = [line for line in lista_peliculas if patron_pelicula.search(line)]

print("Películas producidas antes de 2002:")
for p in peliculas_antes_2002:
    print(p)
```

✓ 0.0s Python

```
Películas producidas antes de 2002:
a. The Shawshank Redemption (1994)
b. The Godfather (1972)
c. The Godfather: Part II (1974)
d. 2001: A Space Odyssey (1968)
e. The Good, the Bad and the Ugly (1966)
f. Angry Men (1957)
g. Schindler's List (1993)
i. Fight Club (1999)
j. 2010: The Year We Make Contact (1984)
k. 101 Dalmatians (1996)
```

Figura 2.6: Resultado del ejercicio 6

Ejercicio 7: Recetas con variaciones de 'chocolate'

```
patron_chocolate = re.compile(r'(?i)c+h+o+c+o+l+a+t+e+')

recetas_con_chocolate = [receta for receta in recetas if patron_chocolate.search(receta)]

print("\nRecetas que contienen la palabra 'chocolate' (con variaciones):")
for receta in recetas_con_chocolate:
    print(receta)
```

[8] ✓ 0.0s Python

```
Recetas que contienen la palabra 'chocolate' (con variaciones):
Cake 1: sugar, flour, cocoa powder, baking powder, baking soda, salt, eggs, milk, vegetable oil, vanilla extract, chocolATE chip.
Cake 3: dark chocolate cake mix, instant CHOCOLATE pudding mix, sour cream, eggs, vegetable oil, coffee liqueur.
Cake 5: gingersnap cookies, chopped pecans, butter, cream cheese, sugar, vanilla extract, eggs, canned pumpkin, cinnamon, CHOColate.
Cake 6: flour, baking soda, sea salt, butter, white sugar, brown sugar, eggs, vanilla extract, Chooocolate chips, canola oil.
```

Figura 2.7: Resultado del ejercicio 7

Ejercicio 8: Inserción de comas en números grandes

```
patron_poblacion = re.compile(r"(\d)(?=(\d{3})+(?!\d))")

poblaciones_con_comas = [patron_poblacion.sub(r'\1,', poblacion) for poblacion in lista_poblaciones]

print("Poblaciones con comas:")
for poblacion in poblaciones_con_comas:
    print(poblacion)
```

[9] ✓ 0.0s Python

```
Poblaciones con comas:
a. China 1,361,220,000
b. India 1,236,800,000
c. United States 317,121,000
d. Indonesia 237,641,326
e. Brazil 201,032,714
f. Pakistan 184,872,000
g. Nigeria 173,615,000
h. Bangladesh 152,518,015
i. Russia 143,600,000
```

Figura 2.8: Resultado del ejercicio 8

Ejercicio 9: Simplificación de direcciones IPv6

```
    direcciones_ipv6 = [
        "2001:0db8:0000:0000:0000:ff00:0042:8329",
        "2607:f0d0:1002:0051:0000:0000:0000:0004"
    ]

    print("\nDirecciones IPv6 simplificadas:")
    for direccion in direcciones_ipv6:
        simplificada = simplificar_ipv6(direccion)
        print(f"Original: {direccion}")
        print(f"Sin ceros a la izq: {remover_ceros_izquierda(direccion)}")
        print(f"Simplificada: {simplificada}\n")
[11]  ✓ 0.0s                                                                Python
...
    Direcciones IPv6 simplificadas:
    Original: 2001:0db8:0000:0000:0000:ff00:0042:8329
    Sin ceros a la izq: 2001:db8:0:0:0:ff00:42:8329
    Simplificada: 2001:db8::ff00:42:8329

    Original: 2607:f0d0:1002:0051:0000:0000:0000:0004
    Sin ceros a la izq: 2607:f0d0:1002:51:0:0:0:4
    Simplificada: 2607:f0d0:1002:51::4
```

Figura 2.9: Resultado del ejercicio 9

2.2. Parte 2: Normalización de Textos

2.2.1. Investigación

En el procesamiento de textos en lenguas naturales, existen diversos módulos en Python que ofrecen funcionalidades específicas para tokenización, lematización, stemming y análisis sintáctico. En esta práctica se compararon tres módulos principales: **NLTK**, **spaCy** y **Stanza**. A continuación se describe cada uno y se justifica su selección:

NLTK (Natural Language Toolkit): NLTK es uno de los frameworks más consolidados en el ámbito del procesamiento de lenguaje natural [1]. Entre sus principales ventajas se destacan:

- **Tokenización:** La función `word_tokenize(text, language='spanish')` permite tokenizar textos en varios idiomas, incluyendo español e inglés.
- **Stemming:** Con `SnowballStemmer('spanish')` se puede aplicar un stemmer específico para español, además de contar con algoritmos para inglés (por ejemplo, `PorterStemmer`).
- **Lematización:** Aunque NLTK incluye un lematizador (`WordNetLemmatizer`) para inglés, su capacidad para español es más limitada, por lo que en la implementación no se pudo llevar a cabo esta misma.

A pesar de ser un módulo robusto y flexible, NLTK a veces requiere mayor configuración y combinación de técnicas para lograr una normalización completa.

spaCy: spaCy es una herramienta moderna y eficiente para el procesamiento de textos, la cual ofrece pipelines optimizados para múltiples idiomas [2]. Entre las variantes disponibles para español se encuentran los modelos `es_core_news_sm` y `es_core_news_md`, que difieren en tamaño y nivel de detalle. Para inglés se utiliza generalmente el modelo `en_core_web_sm`. Entre sus ventajas se incluyen:

- **Integración y rendimiento:** spaCy integra de forma nativa tokenización, lematización, y detección de stop words con una alta eficiencia.
- **Calidad de lematización:** Para ambos idiomas, la lematización es robusta, lo que favorece la reducción de la variabilidad léxica.

La elección de spaCy se justifica por su facilidad de uso y la consistencia de los resultados obtenidos sin necesidad de combinar múltiples herramientas.

Stanza: Stanza es una herramienta desarrollada por el grupo de Stanford NLP que ofrece un pipeline completo para muchos idiomas [3]. Para su inicialización, basta con:

- `nlp_stanza = stanza.Pipeline('en', processors='tokenize,lemma', verbose=False)` para inglés.
- `nlp_stanza_es = stanza.Pipeline('es', processors='tokenize,lemma', verbose=False)` para español.

Entre sus características destacan:

- **Estructura modular:** Permite configurar distintos procesadores, lo que facilita la extensión a tareas adicionales si fuera necesario.
- **Resultados estructurados:** La salida de Stanza es muy consistente, lo que resulta útil para análisis posteriores.

Aunque Stanza puede requerir pasos adicionales (como la aplicación de un stemmer en el caso de español para equiparar los resultados con los de NLTK), su rendimiento en lematización y tokenización resulta muy competitivo.

Selección y Justificación En el código de esta práctica se optó por comparar los tres módulos en lugar de uno sólo como se requería originalmente en la prática; esto para aplicar un enfoque basado en el método científico, evaluando de manera simultánea la eficiencia y la calidad en la tokenización y normalización de textos en inglés y español. Esta comparación nos permitió:

- Contrastar la rapidez de procesamiento entre NLTK, spaCy y Stanza.
- Evaluar la diversidad y el número de tokens obtenidos antes y después de aplicar técnicas de normalización.
- Analizar la calidad de la lematización y el stemming en cada módulo.

- Tener validez empírica sobre los módulos a usar en futuras prácticas y que funcionen mejor para cada caso específico

La combinación de estos análisis nos ofreció una visión integral que respalda la selección de spaCy y Stanza para tareas donde la eficiencia y la precisión en la normalización son prioritarias, mientras que NLTK sigue siendo valioso por su flexibilidad y amplia comunidad de usuarios.
En resumen, la integración de **NLTK**, **spaCy** y **Stanza** en este estudio responde a la necesidad de aplicar un análisis comparativo riguroso, permitiendo identificar las fortalezas y limitaciones de cada herramienta en el contexto del procesamiento de textos multilingües.

2.2.2. Metodología y Pipeline de Normalización

El proceso de normalización de textos se desarrolló siguiendo un pipeline estructurado que permite homogeneizar el contenido y reducir la variabilidad léxica. Este pipeline se compone de los siguientes pasos, aquí se aplican los procesos sugeridos por parte de la práctico, además de algunos otros indicados como adicionales:

1. **Conversión a minúsculas (Adicional):** Se transforma todo el texto a minúsculas para garantizar la uniformidad. Esto elimina diferencias entre mayúsculas y minúsculas que pueden generar duplicidad en los tokens (por ejemplo, *Casa* vs. *casa*).

2. **Remoción de dígitos (Adicional):** Se eliminan los dígitos, ya que en muchos casos no aportan información semántica relevante y pueden introducir ruido en el análisis de frecuencias y en la lematización.

3. **Remoción de puntuación (Adicional):** Se suprime la puntuación, la cual en algunos módulos se puede gestionar de forma similar a la eliminación de *stop words*. Esta acción contribuye a la reducción de ruido y facilita la tokenización.

4. **Tokenización:** Se divide el texto en unidades lingüísticas (tokens) utilizando las funciones específicas de cada módulo (por ejemplo, `word_tokenize` en NLTK, el objeto `doc` en spaCy o el pipeline de Stanza). Este paso es fundamental para segmentar el texto y trabajar en unidades de análisis.

5. **Remoción de *Stop Words*:** Se eliminan palabras de poco significado (como artículos, preposiciones, etc.) que, a pesar de ser frecuentes, no aportan valor semántico al análisis. Cada módulo ofrece un listado propio de *stop words* (por ejemplo, `stopwords.words('english')` en NLTK o el atributo `is_stop` en spaCy).

6. **Aplicación de Stemming y Lematización:** Se aplica stemming (reducción de palabras a su raíz) y lematización (reducción a la forma canónica) para minimizar la variabilidad derivada de la conjugación y declinación de los términos. En NLTK se utilizan `PorterStemmer` para inglés y `SnowballStemmer` para español, mientras que

spaCy y Stanza proporcionan funciones de lematización integradas. Esta doble técnica permite comparar la efectividad de cada método en la reducción de la dispersión léxica.

7. **Filtrado de tokens no alfabéticos y de longitud inferior a 3 caracteres (Adicional):** Se eliminan aquellos tokens que contengan caracteres no alfabéticos o que tengan una longitud inferior a 3, con el fin de descartar símbolos, abreviaturas o tokens irrelevantes que podrían distorsionar el análisis de frecuencias y la representación semántica del corpus.

Justificación de Calidad: El orden de ejecución en este pipeline se fundamenta en la necesidad de partir de un texto lo más homogéneo posible. La conversión a minúsculas y la remoción de dígitos y puntuación crean una base limpia, lo que facilita una tokenización precisa. La eliminación de *stop words* reduce el ruido, permitiendo que las técnicas de stemming y lematización trabajen sobre tokens relevantes, y finalmente, el filtrado de tokens no significativos asegura que el análisis se centre en términos que aportan valor semántico. Esta metodología no solo mejora la calidad de la normalización sino que también permite comparar de forma efectiva los resultados obtenidos con los distintos módulos (NLTK, spaCy y Stanza), validando la robustez de cada enfoque en el tratamiento de textos multilingües.

Figura 2.10: Diagrama del pipeline de normalización de textos.

2.2.3. Análisis Exploratorio y Resultados

Se realizó un análisis exhaustivo de los textos en inglés y en español, tanto en su versión original (pre-normalización) como luego de aplicar el pipeline de normalización. Para ello se evaluaron diferentes métricas utilizando las tres librerías: NLTK, spaCy y Stanza. Las métricas consideradas fueron:

- **Tiempos de Tokenización:** Tiempo requerido para segmentar el texto original.
- **Total de Tokens (Pre-Normalización):** Número total de tokens obtenidos tras la tokenización del texto original.
- **Tokens Únicos (Pre-Normalización):** Número de tokens distintos presentes en el texto original.
- **Top 15 Tokens (Pre-Normalización):** Distribución de frecuencia de los 15 tokens más comunes.

- **Tiempos de Normalización:** Tiempo requerido para aplicar el pipeline completo de normalización.
- **Total de Tokens y Tokens Únicos (Post-Normalización):** Métricas similares a las anteriores, obtenidas sobre el texto normalizado.
- **Top 15 Tokens (Post-Normalización):** Histogramas de los 15 tokens más frecuentes tras la normalización.

Resultados en Inglés

Tiempos de Tokenización:

- NLTK: 0.0223 s (374 tokens, 212 únicos)
- spaCy: 0.0744 s (381 tokens, 216 únicos)
- Stanza: 0.4294 s (374 tokens, 215 únicos)

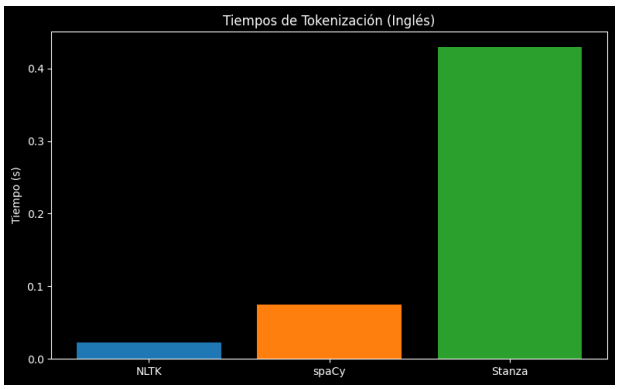

Figura 2.11: Tiempos de tokenización, inglés

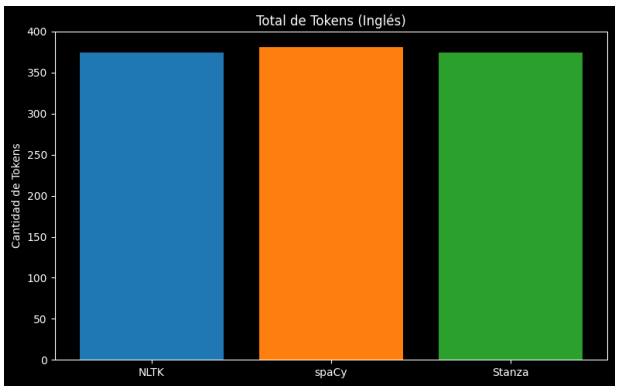

Figura 2.12: Total de tokens, inglés

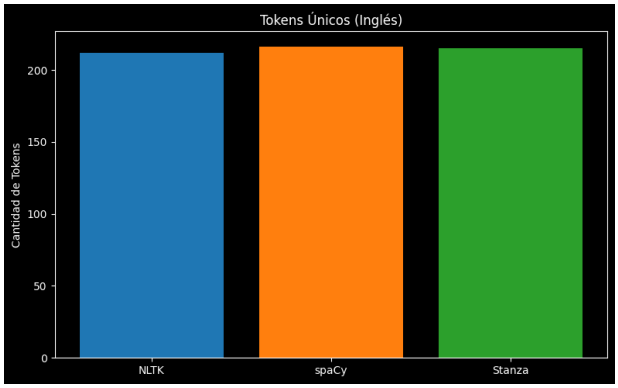

Figura 2.13: Total de tokens únicos, inglés

Tokenización - Top 15 Tokens:

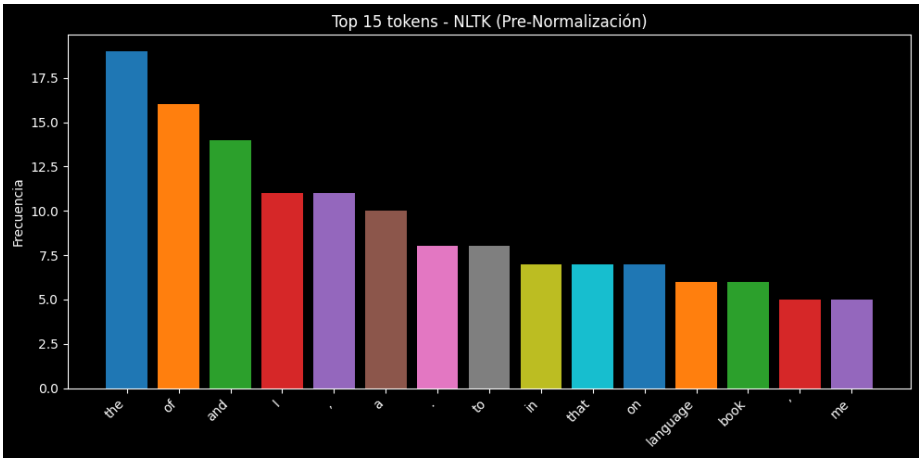

Figura 2.14: 15 tokens - NLTK

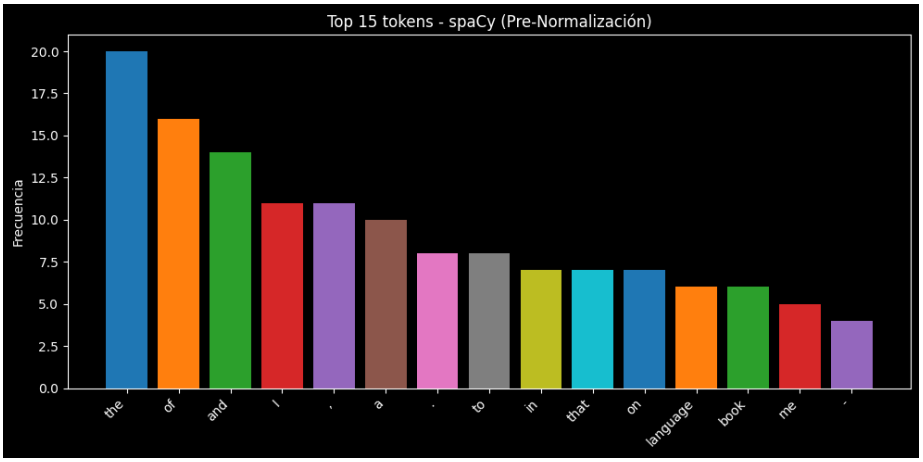

Figura 2.15: 15 tokens - Spacy

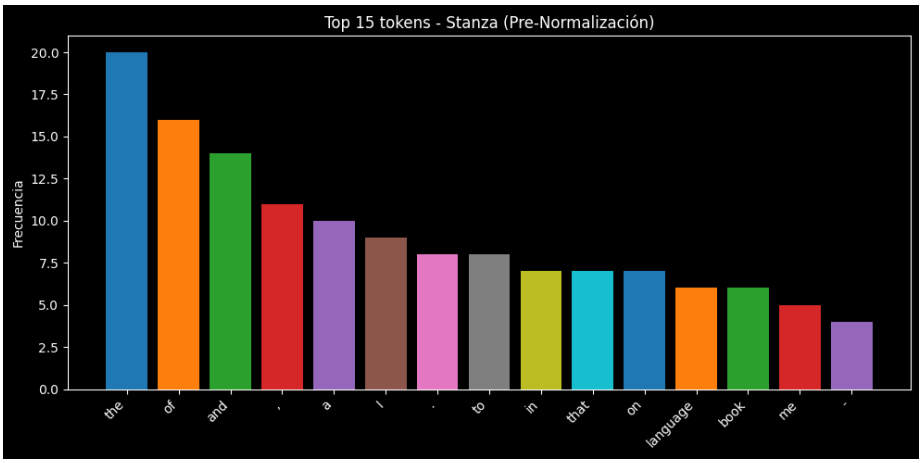

Figura 2.16: 15 tokens - Stanza

Tiempos de Normalización:

- NLTK: 3.5063 s (172 tokens, 136 únicos)
- spaCy: 0.0619 s (160 tokens, 119 únicos)
- Stanza: 0.0450 s (171 tokens, 133 únicos)

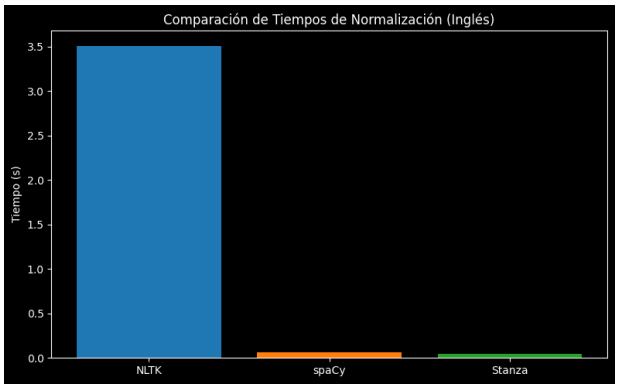

Figura 2.17: Tiempos de normalización en Inglés

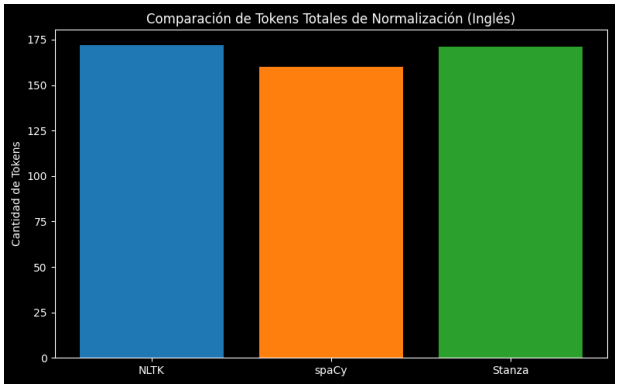

Figura 2.18: Total de tokens post normalización, inglés

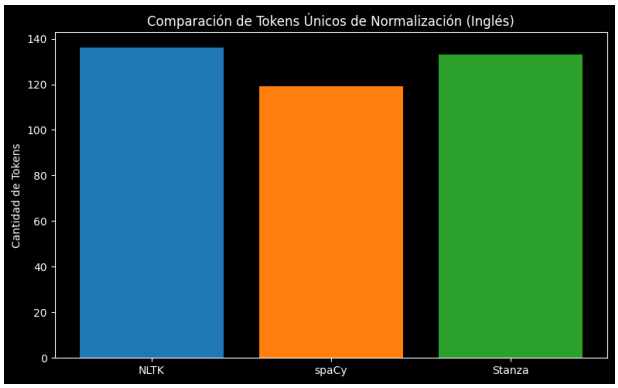

Figura 2.19: Total de tokens únicos post normalización, inglés

Normalización - Top 15 Tokens:

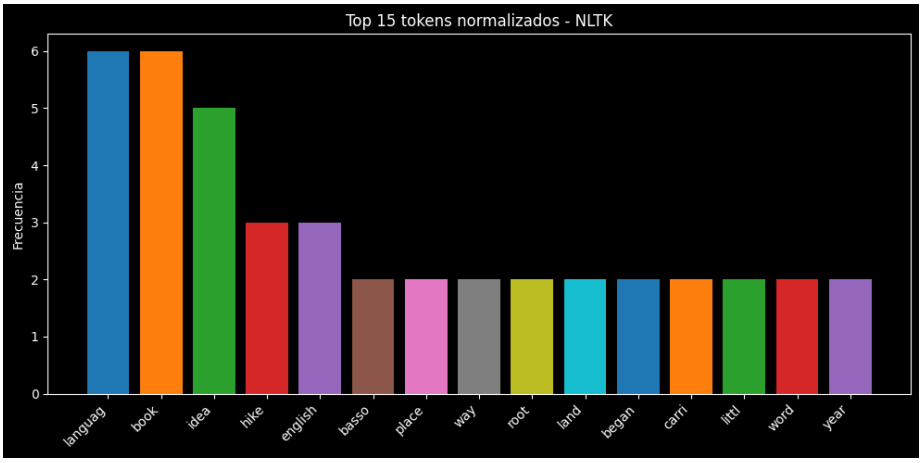

Figura 2.20: 15 tokens post normalización - NLTK

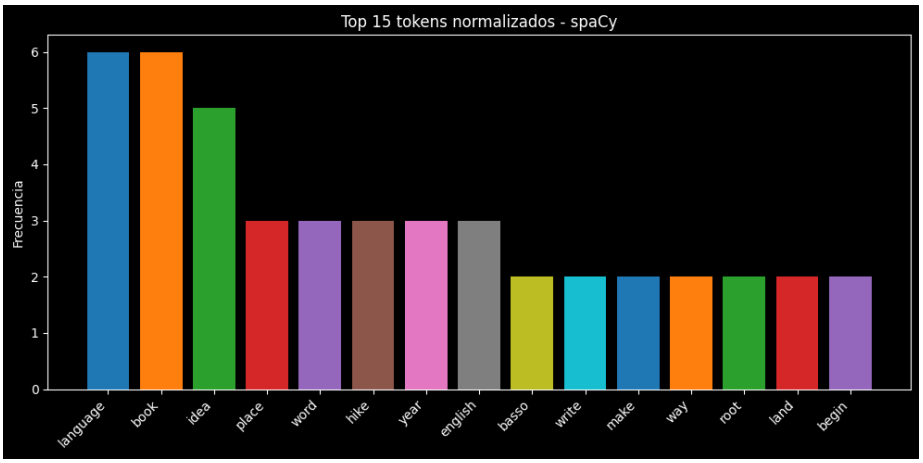

Figura 2.21: 15 tokens post normalización - Spacy

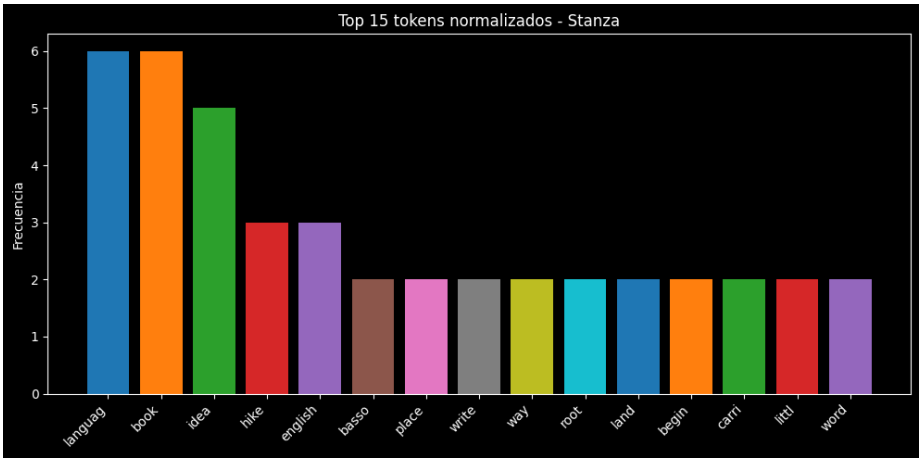

Figura 2.22: 15 tokens post normalización - Stanza

Resultados en Español

Tiempos de Tokenización:

- NLTK: 0.0529 s (373 tokens, 195 únicos)
- spaCy: 0.0726 s (377 tokens, 198 únicos)
- Stanza: 0.1448 s (370 tokens, 196 únicos)

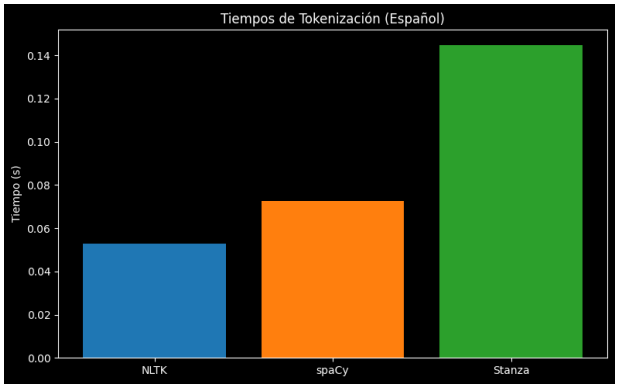

Figura 2.23: Tiempos de tokenización, español

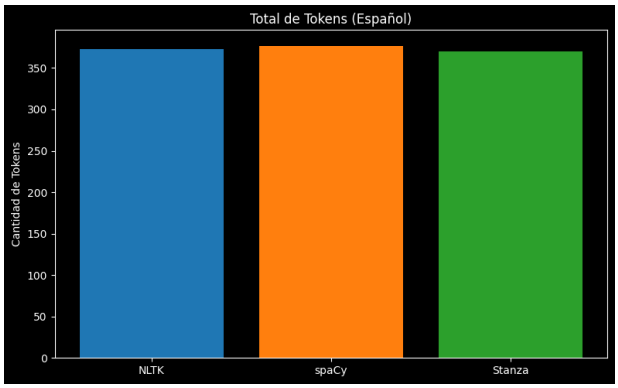

Figura 2.24: Total de tokens, español

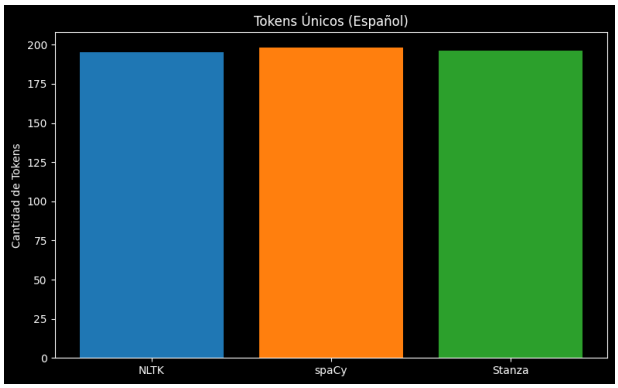

Figura 2.25: Total de tokens únicos, español

Tokenización - Top 15 Tokens:

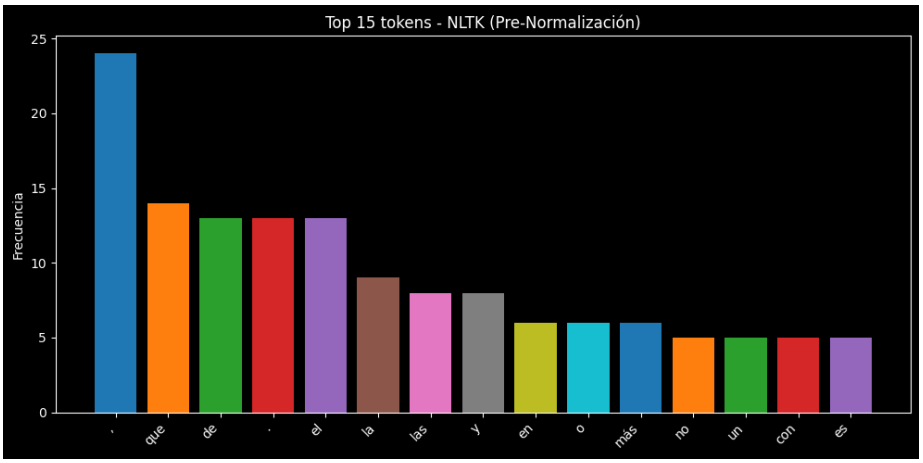

Figura 2.26: 15 tokens en español - NLTK

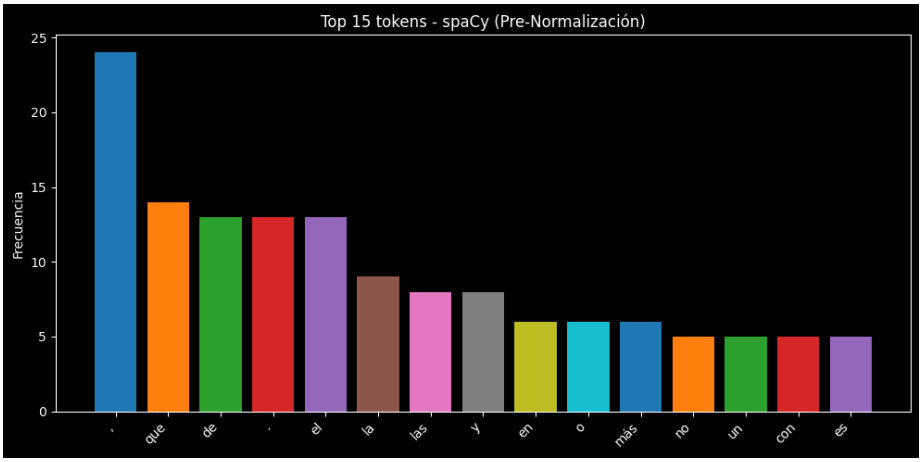

Figura 2.27: 15 tokens en español - Spacy

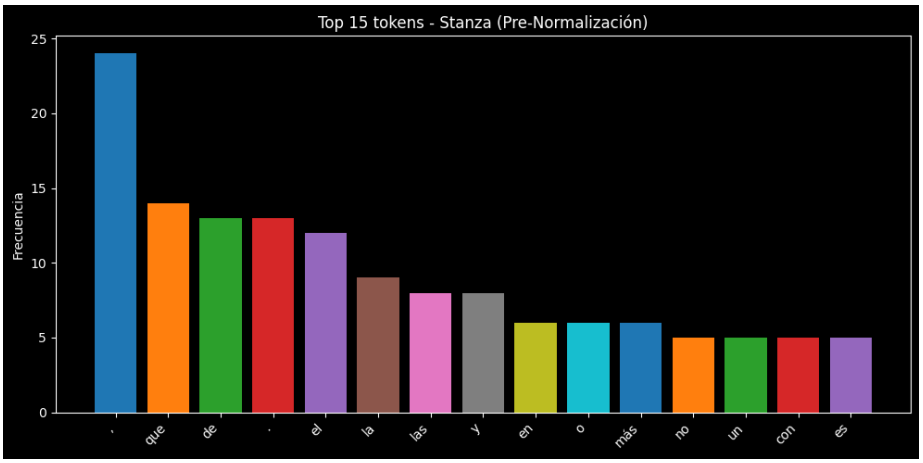

Figura 2.28: 15 tokens en español - Stanza

Tiempos de Normalización:

- NLTK: 0.0034 s (158 tokens, 124 únicos)
- spaCy: 0.0495 s (135 tokens, 110 únicos)
- Stanza: 0.0830 s (172 tokens, 126 únicos)

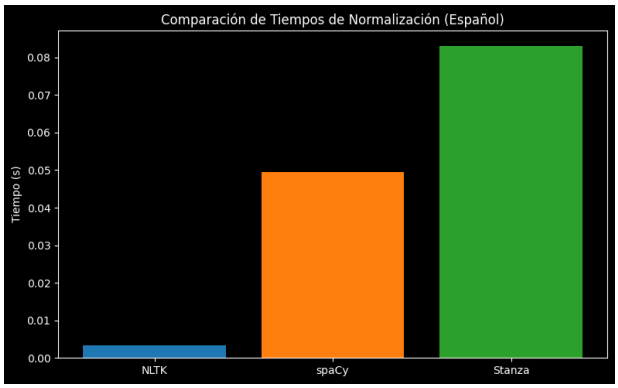

Figura 2.29: Tiempos de normalización en Español

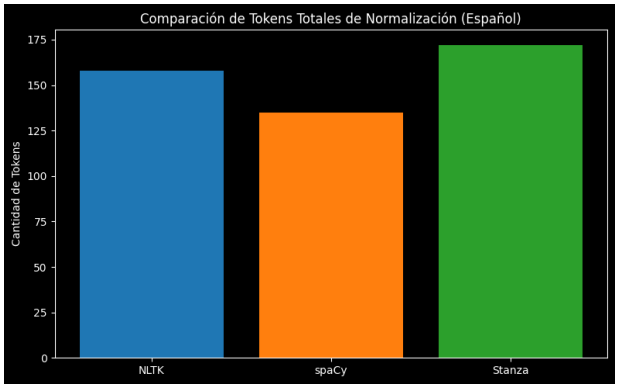

Figura 2.30: Total de tokens post normalización, español

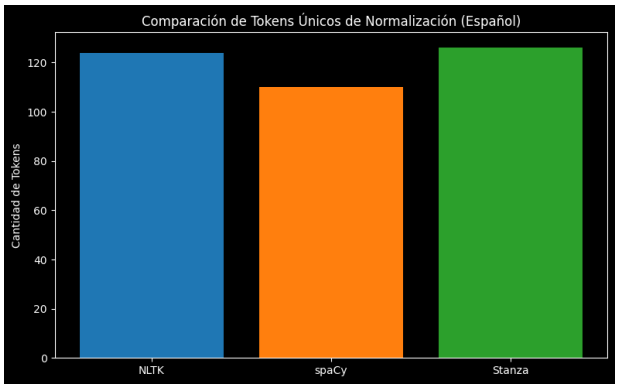

Figura 2.31: Total de tokens únicos post normalización, español

Normalización - Top 15 Tokens:

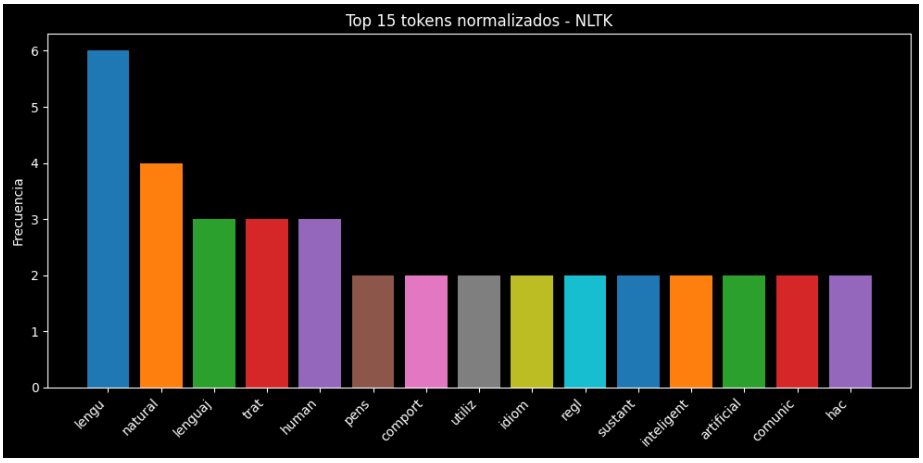

Figura 2.32: 15 tokens post normalización español - NLTK

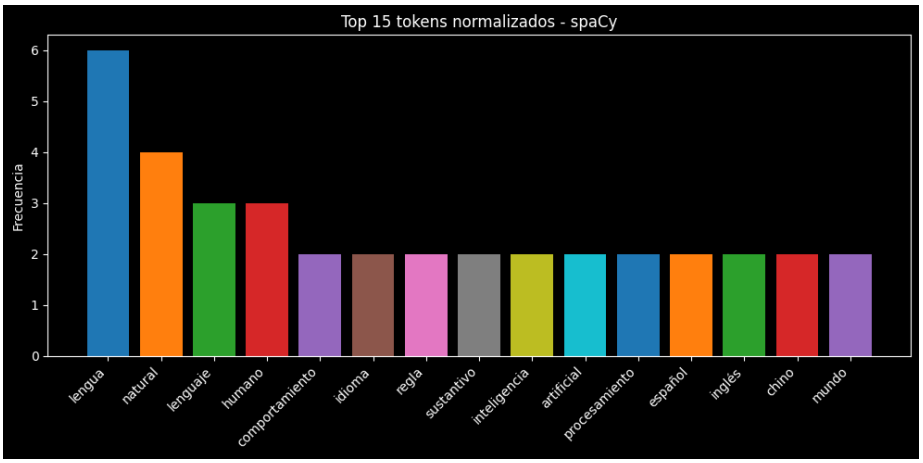

Figura 2.33: 15 tokens post normalización español - Spacy

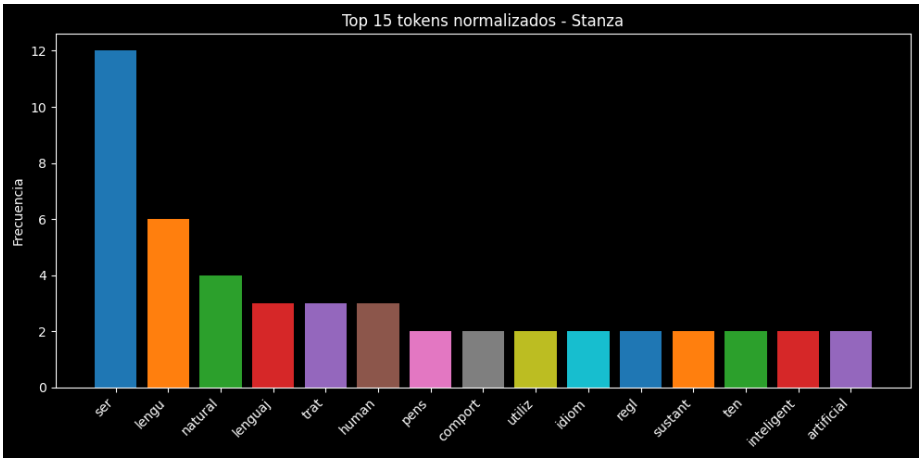

Figura 2.34: 15 tokens post normalización español - Stanza

Observaciones en los Resultados

En Inglés: Los resultados muestran que, en términos de tokenización, spaCy y Stanza generan un número similar de tokens y tokens únicos, mientras que NLTK presenta una ligera variación. En cuanto a la normalización, spaCy y Stanza obtienen textos más homogéneos (menos tokens y tokens únicos) en comparación con NLTK, lo cual indica una mayor efectividad en la reducción de la variabilidad léxica. Los histogramas del Top 15 tokens post-normalización revelan que las técnicas de lematización y stemming han concentrado la frecuencia en términos relevantes como `language`, `book` e `idea`.

En Español: Los tiempos de tokenización son comparables entre los módulos, aunque Stanza presenta un procesamiento ligeramente más lento. Para la normalización, NLTK muestra tiempos muy bajos, pero spaCy y Stanza ofrecen un balance entre eficiencia y calidad, reduciendo significativamente la cantidad de tokens y tokens únicos. Un hallazgo destacable es que, en el caso de Stanza, el token más frecuente post-normalización es `ser` (con una frecuencia de 12). Esto puede deberse a la forma en que el lematizador de Stanza

procesa ciertas conjugaciones verbales en español, dejando el verbo en su forma infinitiva `ser` en lugar de mantener formas conjugadas o aplicar un stemming más agresivo. Aunque esta situación puede interpretarse como una consecuencia de la robustez del análisis morfosintáctico de Stanza, también se recomienda revisar y, de ser necesario, complementar la normalización con un filtro adicional que verifique la relevancia de verbos altamente frecuentes en contextos específicos. En general, este comportamiento es correcto, pero debe interpretarse en función del objetivo del análisis semántico posterior.

En conclusión, la comparación entre los tres módulos confirma que, si bien existen diferencias en tiempos de procesamiento y en la cantidad de tokens generados, todos logran mejorar la calidad del texto para un análisis posterior. La elección de spaCy o Stanza puede depender de la necesidad de balancear entre eficiencia (tiempos de procesamiento) y precisión en la lematización, mientras que NLTK sigue siendo una opción flexible y robusta para ciertas tareas.

2.2.4. Discusión y Conclusiones

El análisis comparativo realizado entre las tres librerías (NLTK, spaCy y Stanza) permitió evidenciar diferencias tanto en términos de rendimiento como en la calidad de la tokenización y normalización de textos en inglés y en español. A continuación, se presentan los hallazgos críticos y se discute el impacto de cada técnica aplicada en el documento final.

Comparación Pre y Post Normalización

La aplicación del pipeline de normalización mostró resultados consistentes en ambos idiomas:

- **Reducción del Número de Tokens:** En ambos idiomas se observó que el total de tokens y la cantidad de tokens únicos disminuyeron tras aplicar la normalización. Esta reducción se debe a la eliminación de stop words, la consolidación de variantes morfológicas a través de la lematización y el stemming, y la eliminación de tokens irrelevantes (como aquellos que contienen dígitos, puntuación o tienen menos de 3 caracteres). Este efecto de homogeneización es beneficioso para posteriores análisis semánticos, ya que minimiza la dispersión léxica y concentra la frecuencia en términos de mayor relevancia.

- **Concentración de Frecuencias:** Los histogramas de los 15 tokens más frecuentes revelaron que, tras la normalización, se evidenció una mayor concentración de frecuencias en ciertos términos claves (por ejemplo, `language`, `book` e `idea` en inglés y `lengua`, `natural` en español). Esto sugiere que la normalización mejora la capacidad de identificar los conceptos centrales del corpus.

Efectividad de las Técnicas Aplicadas

El pipeline de normalización se compuso de seis tareas ordenadas estratégicamente para maximizar la calidad del documento resultante:

1. **Conversión a Minúsculas, Remoción de Dígitos y Puntuación:** Estos pasos previos crean una base homogénea y reducen la variabilidad que pueden generar las diferencias de mayúsculas y minúsculas o la presencia de caracteres irrelevantes. Su impacto es fundamental para asegurar que la tokenización sea precisa.

2. **Tokenización y Eliminación de Stop Words:** La segmentación del texto en tokens, seguida de la eliminación de palabras de poco significado, reduce el ruido y facilita el análisis de frecuencia. Cada librería ofrece sus propios listados de stop words, siendo estos más completos y específicos en spaCy y Stanza para ciertos idiomas.

3. **Stemming y Lematización:** La aplicación simultánea de técnicas de stemming y lematización es crucial para consolidar las distintas formas verbales y nominales en su forma base. Sin embargo, se observó que NLTK carece de un lematizador robusto para español, lo que lo posiciona en desventaja para este idioma en comparación con spaCy y Stanza, cuyos modelos permiten una normalización más efectiva.

4. **Filtrado de Tokens No Alfabéticos y Tokens Cortos:** Este paso adicional elimina tokens que pueden introducir ruido (por ejemplo, símbolos o abreviaturas), lo que contribuye a un corpus más depurado y adecuado para análisis semántico.

La secuencia en que se aplicaron estas técnicas garantiza que cada paso trabaje sobre un texto previamente depurado, maximizando la calidad final del documento. En este sentido, la normalización no solo reduce el tamaño del corpus en términos de número de tokens, sino que también mejora la relevancia de los términos, facilitando la identificación de conceptos clave.

Comparación de Rendimiento y Recomendaciones por Escenario

Una comparación crítica de los tiempos de procesamiento y la calidad de la normalización sugiere lo siguiente:

Módulo	Tokenización	Normalización	Recomendación
NLTK	Rápido en corpus cortos	Más lento en inglés; limitado en lematización para español	Ideal para corpus pequeños en inglés, pero menos robusto en español
spaCy	Buen rendimiento, mayor consistencia	Alta eficiencia en ambos idiomas	Recomendado para grandes volúmenes y análisis multilingües
Stanza	Más lento en tokenización	Muy eficiente en normalización; ofrece resultados consistentes	Útil cuando se requiere una lematización robusta, especialmente en español

Tabla 2.1: Resumen comparativo de rendimiento y calidad entre NLTK, spaCy y Stanza.

Caso Particular: Consolidación del Verbo `ser` en Stanza

Un hallazgo interesante fue la alta frecuencia del token `ser` en el análisis de normalización en español utilizando Stanza. Este resultado se explica por el proceso de lematización de Stanza, el cual transforma diversas formas conjugadas del verbo (por ejemplo, *es*, *soy*, *fue*, *siendo*, etc.) en su forma infinitiva `ser`. Esta consolidación es adecuada, ya que reduce la dispersión léxica y facilita el análisis semántico al unificar las variantes morfosintácticas en un solo token.

Cabe destacar que este comportamiento se observa predominantemente en Stanza, y no en las otras librerías, debido a las siguientes razones:

- **Pipeline de Lematización de Stanza:** Stanza utiliza un lematizador entrenado sobre corpus anotados que optimiza la reducción de la variabilidad morfosintáctica. Como resultado, todas las formas conjugadas del verbo se consolidan en la forma canónica `ser`. Esta estrategia permite una homogeneización muy eficaz, lo que resulta en una frecuencia elevada del token `ser`.

- **Comparación con NLTK y spaCy:** Aunque la lista de *stop words* empleada por Stanza y NLTK es similar — e incluso se solapan en muchos casos— la transformación de las formas verbales es independiente del filtrado de *stop words*. NLTK, al carecer de un lematizador robusto para español, se apoya en técnicas de stemming (por ejemplo, `SnowballStemmer`) que no agrupan de manera tan completa las variantes del verbo. Por su parte, spaCy ofrece un lematizador para español (mediante el modelo `es_core_news_sm`), pero en su caso la unificación de las formas conjugadas no es tan agresiva como en Stanza, lo que mantiene cierta variabilidad en la salida.

En síntesis, la consolidación de todas las formas conjugadas a `ser` en Stanza se debe a su sofisticado proceso de lematización, que logra unificar de forma efectiva las distintas formas verbales. Este comportamiento es deseable para reducir la dispersión léxica, aunque en escenarios donde la diferenciación contextual de verbos comunes pueda ser relevante, se podría considerar la aplicación de filtros adicionales o ajustes en el proceso de normalización.

Conclusiones Finales

Este análisis respalda la elección de combinar y comparar estos módulos para aprovechar las fortalezas de cada uno, permitiendo seleccionar la herramienta más adecuada según el tamaño del corpus, el idioma y la tarea a realizar.

Capítulo 3

Conclusiones Generales

La presente práctica abordó dos aspectos fundamentales del procesamiento de lenguaje natural: la aplicación de expresiones regulares para la extracción de patrones específicos (Parte 1) y la normalización de textos en inglés y español (Parte 2). A continuación, se sintetizan las conclusiones generales derivadas del análisis y los resultados obtenidos en ambas secciones.

Parte 1: Expresiones Regulares

En esta primera parte se diseñaron y aplicaron nueve ejercicios basados en expresiones regulares, cuyo propósito fue extraer información relevante a partir del archivo de datos (Anexo A). Entre los aspectos más destacados se encuentran:

- La capacidad de identificar patrones específicos en los registros, tales como la búsqueda de secuencias de caracteres (por ejemplo, la presencia de una r"seguida de "g.º patrones complejos que validan formatos numéricos en descripciones de gastos).
- La manipulación de formatos complejos, como la simplificación de direcciones IPv6 y la inserción de separadores en números grandes.
- La validación de la correcta implementación del código mediante la captura de salidas en pantalla, lo que evidencia la efectividad de las expresiones regulares en la extracción y transformación de información.

La ejecución de esta parte de la práctica demuestra que el uso adecuado de expresiones regulares es una herramienta poderosa para filtrar y extraer información en contextos donde se trabaja con datos semi-estructurados. La claridad en la definición de los patrones y la correcta segmentación de los registros han permitido obtener resultados consistentes y precisos.

Parte 2: Normalización de Textos

La segunda parte se centró en la comparación y evaluación de tres librerías ampliamente utilizadas en procesamiento de lenguaje natural: NLTK, spaCy y Stanza. Se implementó

un pipeline de normalización que incluyó:

1. Conversión a minúsculas.
2. Remoción de dígitos y puntuación.
3. Tokenización.
4. Eliminación de *stop words*.
5. Aplicación de técnicas de stemming y lematización.
6. Filtrado de tokens no alfabéticos y de longitud inferior a 3 caracteres.

Los resultados del análisis exploratorio mostraron que, tras la normalización, se consigue una reducción significativa en el número total y en la diversidad de tokens, lo cual contribuye a una mayor homogeneidad del corpus. Esto es fundamental para tareas de análisis semántico y de identificación de términos clave. La comparación entre las librerías reveló que:

- **NLTK** es adecuado para corpus pequeños y ofrece un rendimiento eficiente en inglés, aunque presenta limitaciones en la lematización para español.
- **spaCy** combina eficiencia y precisión, siendo recomendado para grandes volúmenes de datos y análisis multilingües.
- **Stanza** destaca por una robusta capacidad de lematización, especialmente en español, pese a tiempos de tokenización ligeramente superiores.

Además, se observó que la consolidación de formas verbales en español mediante la lematización (ejemplificada con la alta frecuencia del token `ser` en Stanza) refuerza la importancia de aplicar filtros y técnicas de normalización para reducir la variabilidad léxica y mejorar la calidad del análisis.

Síntesis y Recomendaciones

En conjunto, la práctica evidencia la importancia de:

- Aplicar expresiones regulares de forma precisa para extraer información útil a partir de datos semi-estructurados, lo cual es esencial en tareas de preprocesamiento y análisis de datos.
- Diseñar pipelines de normalización robustos que incluyan múltiples técnicas (conversión a minúsculas, eliminación de ruido, tokenización, stop words, lematización y stemming) para obtener corpus homogéneos y facilitar el análisis posterior.
- Evaluar y seleccionar la herramienta adecuada en función del idioma, tamaño del corpus y tarea a realizar, aprovechando las fortalezas específicas de cada módulo (NLTK, spaCy y Stanza).

Finalmente, se concluye que la combinación de estas técnicas y herramientas no solo mejora la calidad del procesamiento de los textos, sino que también permite adaptar el enfoque a las necesidades específicas de cada proyecto. La comparación entre los módulos revela que, aunque cada uno tiene sus ventajas, la integración de sus resultados puede ofrecer una perspectiva más completa y precisa del contenido analizado.

Esta práctica, por tanto, no solo demuestra la aplicabilidad de técnicas de extracción y normalización, sino que también subraya la importancia de un análisis crítico y comparativo en el diseño de soluciones para el procesamiento de lenguaje natural.

Bibliografía

[1] S. Bird, E. Loper, and E. Klein, *Natural Language Processing with Python*. O'Reilly Media Inc., 2009.

[2] "spacy: Industrial-strength natural language processing in python." https://spacy.io/. Consultado en marzo de 2025.

[3] P. Qi *et al.*, "Stanza: A python natural language processing toolkit for many human languages," *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 101–107, 2020.