Práctica 1. Expresiones regulares, exploración y normalización de datos

Desarrollado por Hernández Jiménez Erick Yael
Escuela Superior de Cómputo
Ingeniería en Inteligencia Artificial

Para la materia de Tecnologías de Lenguaje Natural
Impartida por Ituriel Enrique Flores Estrada
6 de octubre de 2024

Práctica 1. Expresiones regulares, exploración y normalización de datos

Resumen

En esta actividad se exploran técnicas de procesamiento de texto mediante el uso de expresiones regulares y procesamiento de lenguaje natural (PLN) en Python. Se abordan tareas como la identificación de patrones en textos, el análisis de *tokens* y normalización de documentos, este caso, en español e inglés; así como la eliminación de elementos no relevantes como *stop words*, seguido de la lematización y *stemming*. Además, se aplican técnicas adicionales de normalización y se comparan los resultados obtenidos antes y después de la normalización para evaluar su impacto. Las bibliotecas utilizadas incluyen *spaCy*, junto con herramientas para el análisis exploratorio de tokens y la visualización de resultados.

Palabras clave: Procesamiento de Lenguaje Natural, PLN, Python, spaCy, expresiones regulares, lematización, *stemming*, *stop words*, normalización de texto, análisis exploratorio, *tokenización*, patrones de texto, tokens.

Introducción

El Procesamiento de Lenguaje Natural (PLN) es una rama de la inteligencia artificial que se dedica a analizar y a manipular el lenguaje humano utilizando herramientas computacionales. Uno de los componentes clave es el uso de expresiones regulares (*RegEx* por su reducción en inglés: *Regular Expressions*), que permiten buscar y manipular patrones de texto de manera eficiente.

Expresiones regulares

Las RegEx son secuencias de símbolos que permiten representar patrones de texto utilizados para buscar y manipular fragmentos específicos de cadenas de caracteres. Se emplean en situaciones comunes como nombres de archivos, identificadores, números de teléfono, direcciones postales o fechas y, a través de caracteres especiales, las expresiones regulares se prestan para construir reglas y localizar patrones en cadenas de texto, mejorando la eficiencia en la manipulación de dato, tanto estructurados como no estructurados[1].

Las expresiones regulares en Python están basadas en el estándar POSIX: un conjunto de caracteres asignados como símbolos y operadores que permiten crear patrones complejos para extraer, validar o modificar fragmentos de texto.

Python incorpora una implementación de *RegEx* a través de la biblioteca "*re"*, la se usará para el desarrollo de esta práctica. Así mismo, para facilitar la compresión del desarrollo de la práctica, a continuación, se agrega una tabla con los caracteres usados junto con su descripción[2] en los anexos al final del documento.

---

[1] (Friedl, 2006). Consúltese los capítulos *Introduction to Regular Expressions* y *The Mechanics of Expression Processing*

[2] (Python Software Foundation, 2024). Para una lista completa de los caracteres usados, consúltese la documentación oficial.

Análisis de documentos

Es el proceso mediante el cual se extraen y procesan datos de lenguaje natural (como textos) para su análisis computacional, facilitando tareas como la traducción automática o la búsqueda de información en grandes volúmenes de datos[3].

La actividad se apoya del uso de la biblioteca *spaCy* ya que proporciona modelos pre entrenados de alta calidad que son fáciles de usar y se adaptan a múltiples idiomas[4], lo que simplifica el flujo de trabajo al eliminar la necesidad de configuración manual. Esto hace que spaCy sea una opción más robusta y práctica para aplicaciones de PLN en comparación con NLTK y otras librerías, una de las herramientas para el procesamiento de lenguaje en *Python*, que ofrece múltiples características para la *tokenización*, lematización, etiquetado gramatical, así como otras operaciones lingüísticas, aunque no puede aplicar *stemming*. A continuación, se describen los conceptos de estos procesos[5].

Tokenización

La tokenización es el proceso de segmentar un texto en unidades más pequeñas llamadas tokens, que pueden ser palabras, frases o símbolos. Estas unidades sirven como base para el análisis de textos ya que facilita su análisis para procesos.

Lematización

Proceso que reduce las palabras a su forma base o lema (ej. "niñas" se convierte en "niña") utilizando reglas gramaticales y diccionarios.

Stemming

Similar a la lematización, pero más rudimentario. Corta las palabras para eliminar sufijos y obtener la raíz, aunque no siempre produce formas canónicas correctas

[3] (UNAM, 2024). Consúltense las notas en la primera sección.
[4] (Explosion, 2024). Para más información consúltese la página oficial.
[5] (UNAM, 2024) Consúltense las secciones 1.1.2, 1.1.5

Stop words

Son palabras de alta frecuencia que no aportan información significativa en el análisis de texto (como artículos, preposiciones, etc.), y suelen eliminarse para reducir la cantidad de datos a procesar

Desarrollo

Parte 1: Expresiones regulares

El propósito es familiarizarse con el uso de expresiones regulares para el análisis y manipulación de texto. A través de la identificación de patrones específicos en cadenas de texto, se aplicarán técnicas de búsqueda y validación esenciales para el procesamiento de documentos. Esta sección también busca desarrollar expresiones regulares eficientes y precisas al trabajar con diferentes ejemplos.

Primero se comenzará cargando el documento del anexo A del archivo de indicaciones original, texto que contiene lo siguiente:

```
Amount:Category:Date:Description
5.25:supply:20170222:box of staples
79.81:meal:20170222:lunch with ABC Corp. clients Al, Bob, and Cy
43.00:travel:20170222:cab back to office
383.75:travel:20170223:flight to Boston, to visit ABC Corp.
55.00:travel:20170223:cab to ABC Corp. in Cambridge, MA
23.25:meal:20170223:dinner at Logan Airport
318.47:supply:20170224:paper, toner, pens, paperclips, tape
142.12:meal:20170226:host dinner with ABC clients, Al, Bob, Cy, Dave, Ellie
303.94:util:20170227:Peoples Gas
121.07:util:20170227:Verizon Wireless
7.59:supply:20170227:Python book (used)
79.99:supply:20170227:spare 20" monitor
49.86:supply:20170228:Stoch Cal for Finance II
6.53:meal:20170302:Dunkin Donuts, drive to Big Inc. near DC
127.23:meal:20170302:dinner, Tavern64
33.07:meal:20170303:dinner, Uncle Julio's
86.00:travel:20170304:mileage, drive to/from Big Inc., Reston, VA
22.00:travel:20170304:tolls
378.81:travel:20170304:Hyatt Hotel, Reston VA, for Big Inc. meeting
1247.49:supply:20170306:Dell 7000 laptop/workstation
6.99:supply:20170306:HDMI cable
212.06:util:20170308:Duquesne Light
23.86:supply:20170309:Practical Guide to Quant Finance Interviews
195.89:supply:20170309:black toner, HP 304A, 2-pack
86.00:travel:20170317:mileage, drive to/from Big Inc., Reston, VA
32.27:meal:20170317:lunch at Clyde's with Fred and Gina, Big Inc.
22.00:travel:20170317:tolls
119.56:util:20170319:Verizon Wireless
284.23:util:20170323:Peoples Gas
8.98:supply:20170325:Flair pens
```

Siendo cadenas de datos separadas por dos puntos ‘:’ donde *amount* corresponde a al precio, *category* a la descripción, *date* a la fecha en formato numérico, y *description* a la descripción del objeto o servicio.

El primer numeral nos solicita encontrar las cadenas que contengan una “r” seguida por una “g”. La “r” y la “g” no necesariamente tienen que estar en posiciones consecutivas. Para ello, generamos la siguiente RegEx:

```
(.*)r(.*)g(.*)$
```

Tal que la expresión se interpreta de la siguiente forma: la cadena puede contener uno o más símbolos antes de una ‘r’, siendo seguida por uno o más símbolos hasta encontrar una ‘g’ precedida por uno o más caracteres hasta el final de línea. El resultado es el siguiente:

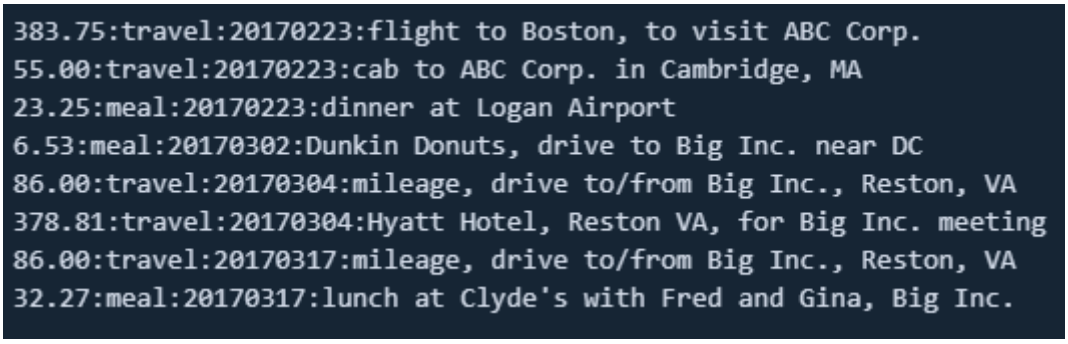

Ilustración 1. Ejecución de la primera expresión regular.

La siguiente expresión encuentra cadenas que describan comidas que cuesten al menos 100.00. Para encontrar el rango numérico posicionado se debe considera que, sea el número $xyy.yy$, x debe pertenecer al rango $[1,9]$, mientras que y al rango $[0,9]$. Además, en la categoría se debe contener la palabra *meal*. Así, la expresión regular es:

```
^[1-9]([0-9]){2,}[.]([0-9]){2}:(meal).*
```

Siendo que, al inicio de la línea, el primer carácter debe ser un dígito del 1 al 9, seguido de dos números (‘{2}’) del rango del 0 al 9, luego un punto ‘.’, otros dos dígitos, dos puntos ‘:’ y finalmente el patrón de la palabra ‘meal’. El resultado es el siguiente:

Ilustración 2. Ejecución de la segunda expresión regular.

Continuando, la siguiente expresión es:

```
^.*(a|A){1}(b|B){1}(c|C){1}.*
```

Tal que al inicio de la línea (‘^’), puede ir cualquier cantidad de cualquier carácter hasta encontrar una sola (‘{1}’) ‘a’ minúscula o mayúscula, seguida de una sola ‘b’ minúscula o mayúscula y finalmente una sola ‘c’ minúscula o mayúscula seguida de cualquier cantidad de cualquier carácter. El resultado es:

```
79.81:meal:20170222:lunch with ABC Corp. clients Al, Bob, and Cy
383.75:travel:20170223:flight to Boston, to visit ABC Corp.
55.00:travel:20170223:cab to ABC Corp. in Cambridge, MA
142.12:meal:20170226:host dinner with ABC clients, Al, Bob, Cy, Dave, Ellie
```

Ilustración 3. Ejecución de la tercera expresión regular.

Para los siguientes dos numerales se tomaron en cuenta dos casos dada la ambigüedad de las indicaciones.

Para el cuarto numeral, en el primer caso se considera la cadena secuencial de la ‘a’ minúscula y los dígitos seguidos sin caracteres intermedios:

```
[:].*(\d+a|a+\d).*$
```

Donde se buscan los dos puntos seguidos por cualquier carácter hasta encontrar un dígito (‘\d’) o más (‘+’) seguido de una ‘a’ minúscula, o una ‘a’ minúscula o más seguido de un dígito, para ser precedido por cualquier carácter hasta el final de la línea. Siguiendo esta expresión, no existe ninguna cadena en el documento que cumpla con estas características.

Por otro lado, si no se considera que deban ser secuenciales con la expresión:

```
.*[:].*[:].*[:].*(\d+.*a|a+.*\d).*$
```

Se obtiene el siguiente resultado:

```
79.99:supply:20170227:spare 20" monitor
127.23:meal:20170302:dinner, Tavern64
1247.49:supply:20170306:Dell 7000 laptop/workstation
195.89:supply:20170309:black toner, HP 304A, 2-pack
```

Ilustración 4. Ejecución de la cuarta expresión regular.

Siguiendo con el quinto numeral, en el primer caso se consideran aquellas cadenas con la subsecuencia ‘di’:

```
.*d{1}.*i{1}.*
```

Dando lo siguiente como resultado:

```
23.25:meal:20170223:dinner at Logan Airport
142.12:meal:20170226:host dinner with ABC clients, Al, Bob, Cy, Dave, Ellie
6.53:meal:20170302:Dunkin Donuts, drive to Big Inc. near DC
127.23:meal:20170302:dinner, Tavern64
33.07:meal:20170303:dinner, Uncle Julio's
86.00:travel:20170304:mileage, drive to/from Big Inc., Reston, VA
23.86:supply:20170309:Practical Guide to Quant Finance Interviews
86.00:travel:20170317:mileage, drive to/from Big Inc., Reston, VA
32.27:meal:20170317:lunch at Clyde's with Fred and Gina, Big Inc.
```

Ilustración 5. Ejecución de la primera versión de la quinta expresión regular.

Mientras que la segunda versión solo considera cadenas con palabras que tengan la subsecuencia ‘di’:

```
^.*d{1}[^ ]*(i){1}.*$
```

Siendo que esta busca después del inicio de cadena tantos caracteres como sean hasta encontrar una única ‘d’ seguida de cualquier carácter que no sea un espacio ‘ ‘ hasta encontrarse con el carácter ‘i’ una vez, luego cualquier carácter hasta el final de la cadena. El resultado es el siguiente:

```
23.25:meal:20170223:dinner at Logan Airport
142.12:meal:20170226:host dinner with ABC clients, Al, Bob, Cy, Dave, Ellie
6.53:meal:20170302:Dunkin Donuts, drive to Big Inc. near DC
127.23:meal:20170302:dinner, Tavern64
33.07:meal:20170303:dinner, Uncle Julio's
86.00:travel:20170304:mileage, drive to/from Big Inc., Reston, VA
86.00:travel:20170317:mileage, drive to/from Big Inc., Reston, VA
```

Ilustración 6. Ejecución de la segunda versión de la quinta expresión regular.

Tras haber trabajado con el mismo documento, en los siguientes numerales se trabajará con documentos distintos, por lo que se colocarán las cadenas junto a los resultados correspondientes.

El numeral 6 solicita encontrar aquellas películas producidas antes del 2002. Esto implica que el número entre paréntesis tenga la siguiente relación: Sean x, y, z dígitos correspondientes al año,$xyyy$ será menor al 2002 si x es menor o igual que 1, y y cualquier dígito del 0 al 9, así como también puede ser $xyyz$ menor si x es 2, y es igual a 0 y z está entre 0 y 1. La expresión, entonces, queda así:

```
.*[(]{1}([0-1]\d{3}|200[0-1])[)]{1}$
```

Y los resultados son:

```
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)
h. The Lord of the Rings: The Return of the King (2003)
i. Fight Club (1999)
j. 2010: The Year We Make Contact (1984)
k. 101 Dalmatians (1996)
```

```
The Shawshank Redemption (1994)
The Godfather (1972)
The Godfather: Part II (1974)
2001: A Space Odyssey (1968)
The Good, the Bad and the Ugly (1966)
Angry Men (1957)
Schindler's List (1993)
Fight Club (1999)
2010: The Year We Make Contact (1984)
101 Dalmatians (1996)
```

Ilustración 7. Ejecución y cadenas correspondientes a la expresión regular 6.

El numeral 7 solicita encontrar la palabra “chocolate” en todas sus variaciones con mayúsculas y minúsculas; para ello se debe buscar el patrón ‘chocolate’ ignorando las mayúsculas, esto siendo posible con la bandera `IGNORECASE` que proporciona el método `re`:

```
.*\bchocolate\b.*
```

Y los resultados son los siguientes:

```
a. Cake 1: sugar, flour, cocoa powder, baking powder, baking soda, salt, eggs, milk, vegetable oil, vanilla extract, chocolATE chip.
b. Cake 2: cream cheese, sugar, vanilla extract, crescent rolls, cinnamon, butter, honey.
c. Cake 3: dark chocolate cake mix, instant CHOCOLATE pudding mix, sour cream, eggs, vegetable oil, coffee liqueur.
d. Cake 4: flour, baking powder, salt, cinnamon, butter, sugar, egg, vanilla extract, milk, chopped walnuts.
e. Cake 5: gingersnap cookies, chopped pecans, butter, cream cheese, sugar, vanilla extract, eggs, canned pumpkin, cinnamon, CHOColate.
f. Cake 6: flour, baking soda, sea salt, butter, white sugar, brown sugar, eggs, vanilla extract, Chooocolate chips, canola oil.
g. Cake 7: wafers, cream cheese, sugar, eggs, vanilla extract, cherry pie filling.
```

```
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.
```

Ilustración 8. Ejecución y cadenas correspondientes a la expresión regular 7.

El numeral 8 solicita manipular el número de poblaciones encontrando las posiciones correspondientes a las comas de separación con expresiones regulares, para esto se aplicaron 3 expresiones regulares que harán la tarea de identificación tras aplicar la coma correspondiente anterior, es decir, que se hará una iteración continua sobre cada cadena y cada expresión hasta ya no encontrar ninguna coincidencia y, en el proceso, agregando las comas correspondientes.

La primera expresión encuentra la posición entre las unidades y las unidades de millar, la segunda la posición entre las unidades de millar y las unidades de millón y, finalmente, la tercera encuentra la posición entre las unidades de millón y las unidades de millar de millón.

Las expresiones correspondientes son:

```
(?<=\d)\d{3}$
(?<=\d)\d{3}(,\d{3}){1}$
(?<=\d)\d{3}(,\d{3}){2}$
```

Leyendo las expresiones de derecha a izquierda, la primera busca tres dígitos al final de la cadena y que sean precedidos por un dígito sin agregar ese dígito a la posición (‘(?<=\d)’); la segunda busca el patrón de 3 dígitos precedidos de una coma una sola vez (‘{1}’), tres dígitos y que sean precedidos de un dígito; y la tercera buscará dos veces exactos el patrón de la coma antes de buscar los 3 dígitos antes y que sea precedido por un dígito. Esta operación de precedencia (o vistazo positivo) garantiza que no se agregue una coma a aquellos números que no la necesiten al inicio. Los resultados son los siguientes:

Ilustración 9. Ejecución y cadenas correspondientes al numeral 8.

Finalmente, para el numeral 9, se solicita la manipulación de cadenas que correspondan a direcciones en el formato IPv6 extendido para simplificarlos. En este caso, el documento no es provisto, pero sí se incluyen un par de ejemplos; mismos que se usarán para probar la funcionalidad del código.

Sea una dirección IPv6 con caracteres pertenecientes al alfabeto hexadecimal; la dirección se divide en grupos de 4 caracteres separados por dos puntos (‘:’). Para simplificar, no es necesario mantener los ceros con los que inicia cada bloque y, por

tanto, ningún bloque que contenga en su totalidad ceros. Por ejemplo, la dirección proporcionada como ejemplo es el siguiente:

```
2001:0db8:0000:0000:0000:ff00:0042:8329
```

Y su simplificación final es la siguiente:

```
2001:db8::ff00:42:8329
```

Para completar el cometido, es necesario eliminar primer los ceros con los que inician los bloques, por lo que se usa la siguiente expresión:

```
(?<=:)0
```

Tal que se buscarán aquellos ceros que sean precedidos por los dos puntos y mismos que, iterativamente, se eliminarán hasta que no haya coincidencias. Posteriormente se buscarán los bloques vacíos resultantes de eliminar los ceros innecesarios. La expresión que se usará es:

```
(?<=[^:]:)::
```

Así, encontraremos los caracteres dos puntos que se repitan en consecuencia de la eliminación.

Los resultados son los siguientes:

Ilustración 10. Dirección IPv6 ingresada.

Ilustración 11. Dirección IPv6 simplificada.

Con esto, se termina la primera parte de la práctica. Para más detalles de cómo funciona el código, ingrese al cuaderno Jupyter con el nombre de archivo “Parte-1.ipynb”.

Parte 2: Normalización de textos

El objetivo es aplicar técnicas de procesamiento del lenguaje natural (PLN) con la particularidad de ser desarrollado con Python, y que se centra en el análisis exploratorio de los textos permitiendo comprender la estructura y características del contenido textual.

Además, se busca implementar un flujo de normalización de texto que incluya la eliminación de stop words, lematización y stemming, así como otras técnicas adicionales que mejoren la calidad de los datos para su análisis.

Biblioteca spaCy

La biblioteca *spaCy* permite el procesamiento eficiente de grandes volúmenes de texto y proporciona acceso a representaciones detalladas de documentos a través de su clase "*Doc*" que facilita el manejo y análisis de palabras, frases y oraciones. Esta biblioteca se utiliza en la práctica para llevar a cabo el análisis exploratorio y la normalización de textos en español e inglés, lo que incluye la eliminación de stop words, la lematización y el stemming, con el fin de mejorar la calidad de los datos para su posterior análisis.

Una de las ventajas de esta biblioteca es la existencia de modelos pre entrenados en cada idioma, por lo que no es necesario importar varias librerías a la vez para explorar los documentos en ambos idiomas. En contraste, la librería no cuenta con un método función para el proceso de *stemming*, por lo que se complemento la librería programando una función básica para cumplir con el cometido: la función `stemmer`.

La función `stemmer` recibe como entrada una palabra al que le aplicará el proceso y el idioma en el que está la palabra. El stemming se llevará a cabo mediante dos listas con los sufijos que se eliminarán en cada idioma: -ar, -er, -ir, -ando, -iendo, -ado, -ido, -ción, -es, -mente, -s, -a, -o para el español; y -ing, -ed. -ly, -s, -es, -er, -er, -est, -y para el inglés. La función, entonces, retornará el 'estema' de la palabra.

Cabe menciona que, al ser una función sin muchas restricciones ni verificaciones, detona un error que no es muy compatible con el flujo desarrollado ya que cabe la posibilidad de eliminar palabras cortas cuya terminación es similar a la de los sufijos y que no son nexos ni artículos, tal como se ve en el ejemplo en inglés.

El flujo de normalización es el siguiente:

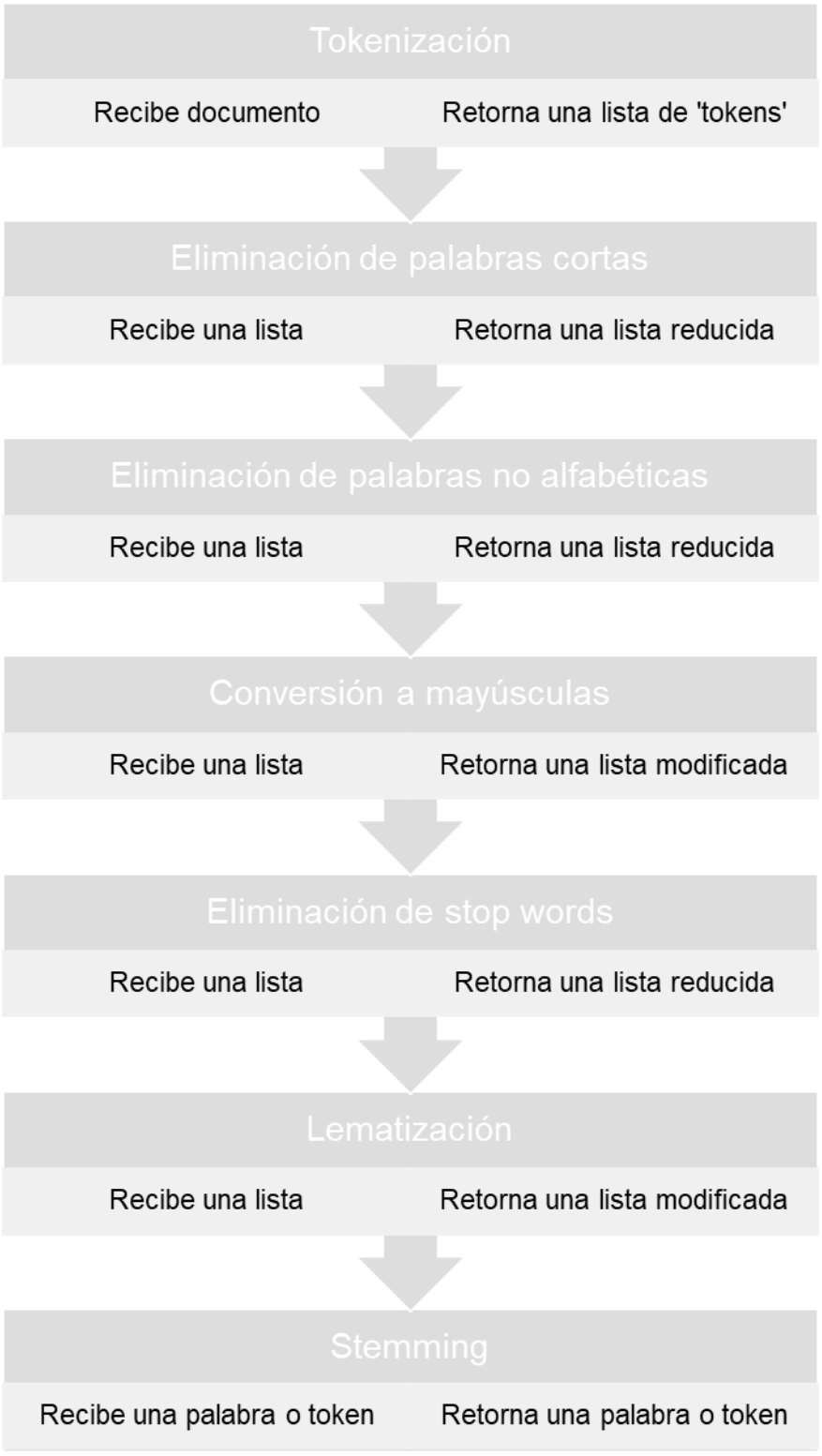

Ilustración 12. Gráfica del flujo de normalización.

El enfoque de este flujo es la reducción de tiempo y el procesamiento de tokens que no son de interés para ahorrar recursos, siendo que se inicia el flujo con una depuración del documento para luego aplicar técnicas más complejas.

Primero se recibe el documento que se tokeniza con la propiedad `text` que extrae el token y lo vuelve una cadena de texto para ser manipulado en los procesos siguientes en la lista en la que se guardará temporalmente a lo largo del flujo. Aunado a ello, se discriminan aquellos tokens que son signos de puntuación o espacios en blanco.

Posteriormente se eliminan las palabras cortas con la función `eliminar_palabras_cortas` que recibe la lista de tokens y la longitud mínima para que permanezcan en la lista, regresando una lista presumiblemente más corta con palabras de 3 o más caracteres en esta aplicación, eliminando artículos y nexos, por ejemplo.

Una vez que se hace la primera depuración, procedemos a la eliminación de palabras que contengan caracteres especiales o números con la función `eliminar_no_alfabeticos` que recibe la lista de tokens y devuelve una lista sin estas palabras no alfabéticas.

Luego se convierten todas las palabras a minúsculas para normalizar el análisis y reducir la cantidad de tokens que sean iguales en su patrón sin considerar combinaciones entre mayúsculas o minúsculas con la función `convertir_a_minusculas`. Esta función retornará una lista con los cambios mencionados.

Tras convertir las palabras a su equivalente en minúsculas, se procede a la eliminación de palabras consideradas *stop words* que pudieran permanecer tras las depuraciones anteriores aprovechando la propiedad `is_stop` de los objetos `doc` de *spaCy* en la función `remover_stop_words` que recibe esta lista de tokens y el modelo del idioma al que pertenecen. Como resultado regresarán una lista de tokens.

Continuando, se lematizan las palabras con lematizar que recibe la lista de tokens y el modelo del idioma al que pertenecen. Esta función aprovecha la propiedad incluida en la biblioteca: `lemmas_`, misma que localiza el lema al que pertenece cada palabra o token para, entonces, regresar una lista con los lemas.

Finalmente se procesa la lista con los estemas para, de ser posible, reducir los tokens a su mínima expresión con la función `stemmer`. Como se mencionaba anteriormente, esta función rudimentaria logra cumplir con el cometido gran parte de los casos, pero en los que no, recorta palabras que, si se itera más de una vez el proceso tal como fue solicitado, se pierden debido al flujo de normalización.

A continuación, se colocará una muestra de los primeros elementos resultantes a la primera normalización de cada texto, tanto en código como en texto.

```
Texto normalizado con stemming en español: ['general', 'pens', 'complejidad', 'lenguaj', 'comportamient
Texto normalizado con stemming en inglés: ['journe', 'language', 'spark', 'discov', 'keith', 'basso', '
```

Ilustración 13. Muestra de primeros elementos en la primera normalización.

Así como los resultados de la normalización iterada 3 veces.

```
Texto normalizado iterativamente en español: ['general', 'p', 'complejidad', 'lenguaj', 'comportamient', 'intuitiv', 'utiliz'
Texto normalizado iterativamente en inglés: ['journe', 'language', 'spark', 'discov', 'keith', 'basso', 'astonish', 'book', '
```

Ilustración 14. Muestra de primeros elementos tras 3 normalizaciones.

Finalmente, como fue solicitado, se mostrarán las gráficas con las palabras más frecuentes en cada documento.

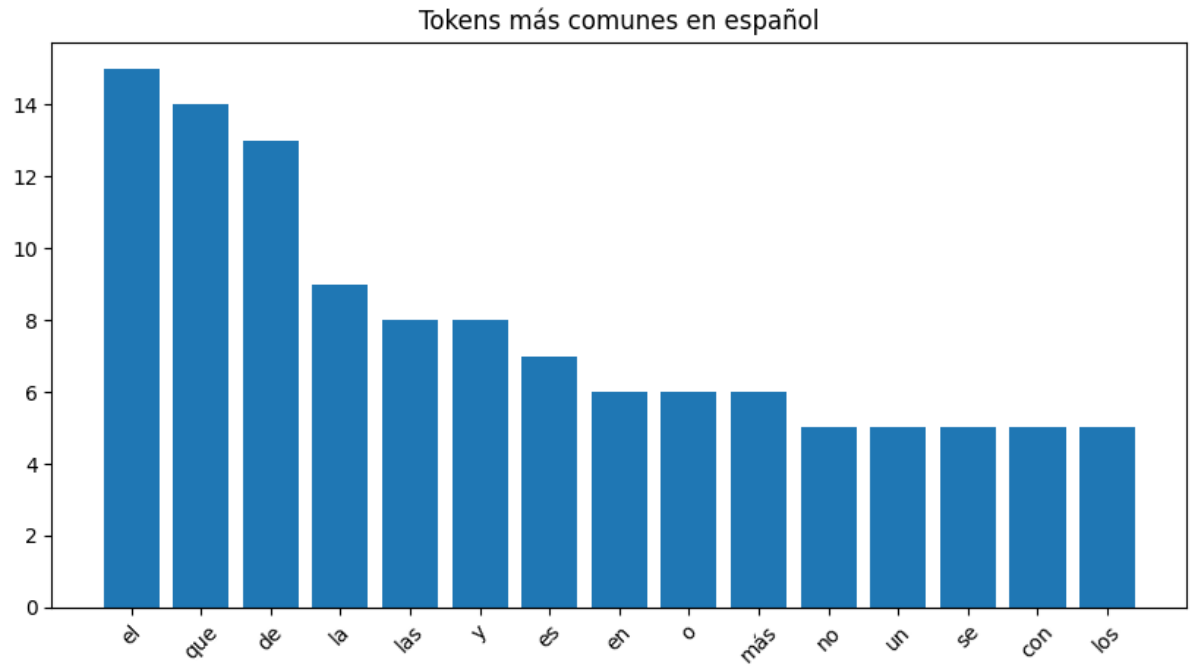

Ilustración 15. Gráfica de las palabras más frecuentes en el documento en español.

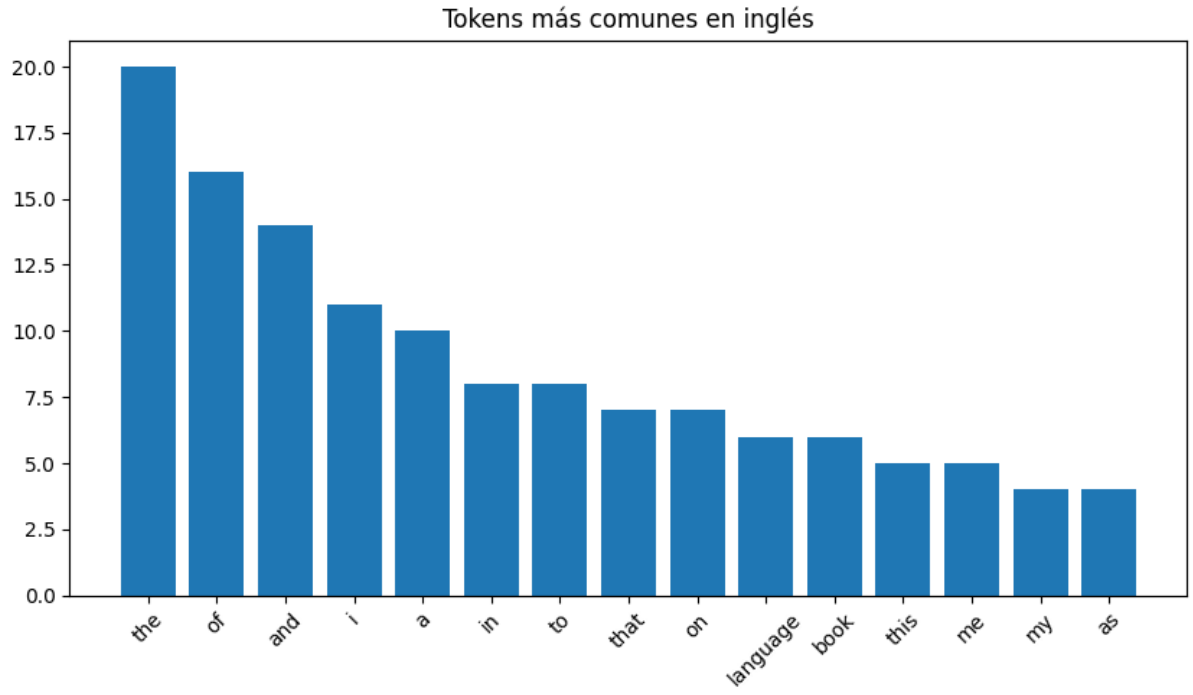

Ilustración 16. Gráfica de las palabras más frecuentes en el documento en inglés.

Con esto se finaliza la segunda parte. Para más detalles de cómo funciona el código, consúltese el cuaderno Jupyter con el nombre de archivo “Parte-2.ipynb”.

Conclusiones

Primera parte

En esta parte de la práctica se puede evidenciar la relativa facilidad que ofrece la búsqueda por expresión regular de elementos y patrones en una cadena de caracteres o documentos, especialmente cuando el texto está normalizado para su procesamiento. Cabe mencionar que, con los resultados obtenidos, al momento de generar expresiones es sustancialmente importante reconocer el patrón deseado y las adecuaciones necesarias para localizarlos en el texto ya que, de lo contrario, su identificación se complica o resulta una expresión poco optimizada y eficaz.

Segunda parte

Con esta sección, y con el flujo propuesto, resultan dos observaciones a destacar. La primera corresponde a la selección correcta de bibliotecas, ya que, si bien por el enfoque de evitar usar varias librerías y la menor cantidad de recursos posibles, la variabilidad en el proceso de análisis de cada idioma hace que la precisión del proceso se vea reducido al involucrar más de un idioma; si bien la función `stemmer` logró cubrir

la carencia de una función integrada, por los resultados en la normalización iterativa es evidente que no es una opción eficiente dado que deteriora los tokens.

Complementariamente, el segundo punto es la construcción de un flujo de normalización adecuado pues impacta enormemente en los resultados que se obtendrán. Por ejemplo, en este flujo la reducción a estemas no resultó tan conveniente debido a la degradación de los tokens que conllevaba, especialmente en el idioma inglés, por lo que sería mejor optar por otras alternativas que no ocasionen este efecto y el resultado sea mejor para este enfoque de eficacia en la normalización de documentos.

Bibliografía

Explosion. (2024). *spaCy*. Obtenido de Linguistic Features: https://spacy.io/usage/linguistic-features

Explosion. (2024). *spaCy*. Recuperado el 10 de 2024, de Doc: https://spacy.io/api/doc

Friedl, J. E. (2006). *Mastering Regular Expressions.* Sebastopol: O'Reilly Media, Inc. Recuperado el 10 de 2024, de https://bmansoori.ir/book/Mastering%20Regular%20Expressions.pdf

Python Software Foundation. (04 de Oct de 2024). *re — Regular expression operations*. Obtenido de re — Regular expression operations: https://docs.python.org/es/3/library/re.html

Universitar Oberta de Catalunya (UOC). (2020). *Introducción a las expresiones regulares.* Catalunya: Universitat Oberta de Catalunya (UOC). Recuperado el 10 de 2024, de https://openaccess.uoc.edu/bitstream/10609/150294/6/ExpressionesRegulares.pdf

Anexos

Tabla 1. Lista y descripción de caracteres usados para *RegEx*

Símbolo o combinación	**Descripción**
.	Coincide con cualquier carácter excepto los saltos de línea
^	Coincide con el inicio de una nueva línea
$	Coincide con el final de una línea
+	Coincide con uno o más repeticiones del patrón o símbolo que le antecede
*****	Coincide con cero o más repeticiones del patrón o símbolo que le antecede
{x}	Busca “x” copias de la expresión que le antecede
{x,y}	Donde “x” es el mínimo y “y” el máximo, esta expresión busca mínimo “x” repeticiones y máximo “y” repeticiones de la expresión regular que le antecede. Colocar los dos argumentos es opcional pero siempre es necesario colocar al menos un argumento y la coma en la posición deseada.
****	Inicia una expresión con caracteres que se incluyen originalmente como operadores para buscarlos como parte de la cadena. Solo escapa un símbolo a la vez
[ ]	Conjunta un grupo de caracteres que pueden ser listados individualmente
|	Equivale a una operación de disyunción, buscará patrones que coincidan con los patrones que le antecedan y le sigan.
( )	Agrupa una expresión concreta y que busca coincidir la expresión exacta
(?<=x)	Busca coincidencias que sean precedidas del patrón “x” sin tener que incluir “x” en la coincidencia
\b	Indica el inicio o el final de una palabra dependiendo de dónde sea colocada en la expresión.
\d	Coincide con cualquier símbolo numérico