# Benemérita Universidad Autónoma de Puebla

***Recuperación de la Información***

Profesora: Maya Carrillo Ruiz

## Práctica:

*Preprocesamiento Colección*


Alumnos:

- Ramírez Varela Axel Daniel
- Bañuelos López Marco Antonio


**Primavera 2021**

Para esta práctica, del enlace: http://ir.dcs.gla.ac.uk/resources/test_collections/cisi/ tenemos una colección de documentos y consultas (1460 y 112 respectivamente), donde todo el texto será preprocesado, es decir tokenizaremos, eliminaremos signos de puntuación, convertiremos a minúsculas eliminaremos palabras vacías y truncaremos. Posteriormente, debemos imprimir las primeras 100 palabras de uno de los documentos de la colección, donde podremos comprobar la ejecución correcta del programa al observar las diferencias entre el texto original y los resultados impresos, además de que podemos guardar en un documento aparte el texto convertido.

**Palabras Clave: Preprocesamiento, colección**

## I. INTRODUCCIÓN

Anteriormente realizamos el preprocesamiento de un único archivo de texto, recordemos que para el análisis correcto del texto tenemos que seguimos una serie de pasos ordenados que consistían en separar en tokens las palabras, eliminar signos de puntuación y convertir a minúsculas, para posteriormente eliminar las palabras vacías y truncar las palabras restantes con el algoritmo de Porter para obtener el “stem” o raíz de las palabras del texto. De esta manera podíamos hacer un análisis que nos permita resumir grandes cantidades de datos en un corto periodo de tiempo. En este caso para la práctica, una colección de datos, en concreto 1460 documentos y 112 consultas deberán ser procesados.

## II. OBJETIVO Y PLANTEAMIENTO DEL PROBLEMA

Actualmente, ya tenemos programados los pasos para realizar el preprocesamiento de un texto, uno de los objetivos de esta práctica es ampliar el análisis por volumen, es decir, tomar toda la colección de documentos y consultas y realizar un análisis equivalente para todo el texto que pueda leer y lo organice en un documento para comprobar que los resultados sean correctos, a continuación, explicaremos cuales son los puntos importantes en el programa para realizar el preprocesamiento de manera correcta.

**III. DESARROLLO EXPERIMENTAL**

La tarea primordial para el preprocesamiento de la colección, consiste en el análisis correcto del texto en sí; obviamente no es útil un análisis si no se realizan los pasos de separación de tokens, eliminar puntuación, etc.

Para el procesamiento de un texto tenemos que realizar 5 pasos

- Separar las palabras en tokens
- Eliminar signos de puntuación
- Convertir todas las palabras a minúsculas
- Eliminar palabras vacías
- Truncar las palabras

Tomemos en cuenta que, para realizar estas prácticas, programaremos en Python apoyándonos en el uso de la librería NLTK, con funciones que nos permiten realizar todas las tareas.

Para el primer paso, separaremos en tokens[1] el texto es decir separar por espacios, una función básica para comenzar a realizar el preprocesamiento, usando la línea de código:

```
words = text[y].split()
```

En el caso para eliminar los signos de puntuación importaremos primero “import re” sirve para operaciones con expresiones regulares, luego la línea que realizara la eliminación es:

```
words= re.split(r'\W+', text[y])
```

la función re.split() nos permite separar y eliminar los signos de puntuación de las palabras que encuentre. La eliminación de signos de puntuación es para evitar que el procesador confunda palabras por ejemplo al terminar un texto, una oración, etc. Al estar los signos junto a la palabra, la computadora lo podría traducir y clasificar como si fueran palabras diferentes,

Para convertir las palabras a minúsculas[2] utilizaremos la línea:

```
words = [word.lower() for word in words]
```

De esta manera con la función lower(), nos permite que la variable words que contiene todo el texto, para todas las mayúsculas que encuentre las convierta a

minúsculas, el objetivo de convertir a minúsculas es por ejemplo para evitar el análisis incorrecto al tomar palabras vacías que estén en mayúsculas estas se encuentren en el análisis final, por ejemplo si un texto comienza con la frase

El perro corre por el campo…

Si no realizamos la conversión, y eliminamos las palabras vacías el resultado podría ser, por ejemplo:

El perro corre campo…

Ahora si consideramos que la mayoría de los párrafos comienzan con un artículo, tendríamos como resultado final una gran cantidad de palabras vacías que no aportan nada y entorpecerían el conteo de datos, igualmente no solo sirve para eliminar eficazmente palabras vacías, también estamos "estandarizando" al texto al tener todas las palabras a minúsculas, si tomamos por ejemplo que un texto menciona la Ciudad de México, y en otra oración menciona ciudad en México, sabemos que tienen distintos significados al referirse tanto a la capital del país, una locación en el mapa específica y también referirse a uno de los estados del país una locación cualquiera del mapa; que tenga la misma palabra no quiere decir que tengan el mismo significado, sin embargo para el programa, no es importante el significado sino cuantas veces se encuentre la palabra para obtener estadísticas o englobar en un grupo una serie de textos que compartan características en común.

El siguiente proceso que es eliminar las palabras vacías[4] es importante resaltar la línea:

nltk_stopwords = set(stopwords.words('english'))

De esta forma estamos especificando que solo eliminaremos palabras vacías en inglés, como dato adicional NLTK tiene la opción para alrededor de 40 idiomas con su lista de palabras vacías; estas palabras son principalmente artículos, pronombres, preposiciones, etc. Es decir, palabras que no cuentan con un significado de importancia o no aportan nada al texto. Podemos encontrar alrededor de 127 palabras[5] consideradas como stopwords o palabras vacías en inglés en la librería de NLTK, El objetivo de eliminar estas palabras es como mencionamos antes porque no aportan nada y al ser datos basura para obtener el corpus del texto.

Finalmente, para truncar palabras[6], se programaron las siguientes líneas

```
for x in range(1,101):
    ww= text_without_stopword[x]
    todo.append(ps.stem(ww))
print(todo[1:100])
```

Al referirnos al truncar palabras, tenemos que entender que es el algoritmo de Porter y el “stem” de las palabras[7] [8]; el “stemming” es un método para reducir una palabra a su raíz o (en inglés) a un stem. De esta forma aumenta en medida el número de documentos que se pueden encontrar con una consulta al relacionar palabras que puedan referirse al mismo tema. Por ejemplo, una consulta sobre "bibliotecas" también encuentra documentos en los que solo aparezca "bibliotecario" porque el stem de las dos palabras es el mismo ("bibliotec"). Aqui es donde entra el algoritmo de Porter cuya función es asegurarse que la forma de una palabra no penalice la frecuencia de estas. Por ejemplo, una palabra puede estar conjugada en cualquier género, número, persona y esta será considerada como un solo término.

Un clásico ejemplo es la frase:

“Aquel es un caballo de la caballería militar, los otros caballos no”

Encontramos entonces 3 palabras que hacen referencia a caballo, el primero menciona a un animal, el segundo a una fuerza de combate y el tercero a el grupo de animales. El algoritmo de Porter entonces remueve los “sufijos comunes morfológicos e inflexionales” de palabras que son diferentes pero que cuentan con una raíz en común, en el ejemplo el stem de las tres palabras es caball. Igualmente sirve para normalizar oraciones, si tomamos como ejemplo dos frases

I was taking a ride in the car.

I was riding in the car.

Ambas frases quieren decir lo mismo, el carro, el tiempo, la persona, la actividad es la misma; sin embargo, las dos frases tienen el mismo significado, pero están escritas de diferente manera para el preprocesamiento no nos interesa como este redactado sino si el contenido es similar para clasificarlo en un mismo grupo o uno diferente.

Ya que describimos como realiza el preprocesamiento de datos tenemos entonces que manejar el análisis para todo el texto, guardarlo en otro archivo, y por comprobación observar la pantalla

De la colección hacemos el análisis de todos los documentos juntos en un solo texto, donde guardamos el texto que inicie en “.W” y terminara cuando encontrase en una nueva línea un “.X” Para entonces solo tomar el texto a procesar y leer línea por línea para trabajarla uno por uno. Este proceso lo llevamos a cabo principalmente en las siguientes líneas:

```
if text[x] == ".W"+"\n":
                    y = x+1
                    words=[]
                    while text[y] != ".X"+"\n":
```

Posteriormente del while realizará todo el análisis con los 5 pasos ya mencionados. Así al ejecutar, el programa realizará dos funciones, imprimirá en pantalla TODAS las oraciones del texto mostrando los 5 pasos antes mencionados en cada oración, y la otra es que en el archivo Q.txt del archivo rar guardará los resultados finales del archivo de texto totalmente convertido, esto para una mejor lectura del texto, el archivo TodosLibros.txt no sufrirá ningún cambio. Completando entonces la ejecución del programa.

El código final junto con los dos documentos TodosLibros.txt y Q.txt han sido puestos en un repositorio de Github: https://github.com/equiporec3/DepartamentalR

## IV. DISCUSIÓN Y RESULTADOS

Como resultado principal tenemos la impresión de pantalla de las primeras 2 oraciones del párrafo separadas mostrando los cinco pasos que mencionamos de tokenización, eliminación de puntuación, conversión, truncar y el stemming; mostrados frase por frase

```
The present study is a history of the DEWEY Decimal

['The', 'present', 'study', 'is', 'a', 'history', 'of', 'the', 'DEWEY', 'Decimal']
['The', 'present', 'study', 'is', 'a', 'history', 'of', 'the', 'DEWEY', 'Decimal', '']
['the', 'present', 'study', 'is', 'a', 'history', 'of', 'the', 'dewey', 'decimal', '']
['present', 'study', 'history', 'dewey', 'decimal', '']
['present', 'studi', 'histori', 'dewey', 'decim', '']

Classification.  The first edition of the DDC was published

['Classification.', 'The', 'first', 'edition', 'of', 'the', 'DDC', 'was', 'published']
['Classification', 'The', 'first', 'edition', 'of', 'the', 'DDC', 'was', 'published', '']
['classification', 'the', 'first', 'edition', 'of', 'the', 'ddc', 'was', 'published', '']
['classification', 'first', 'edition', 'ddc', 'published', '']
['classif', 'first', 'edit', 'ddc', 'publish', '']
```

Posteriormente tenemos para comparación el texto original contra la impresión final del texto con los cambios realizados

```
Comaromi, J.P.
.W
The present study is a history of the DEWEY Decimal
Classification.  The first edition of the DDC was published
in 1876, the eighteenth edition in 1971, and future editions
will continue to appear as needed.  In spite of the DDC's
long and healthy life, however, its full story has never
been told.  There have been biographies of Dewey
that briefly describe his system, but this is the first
attempt to provide a detailed history of the work that
more than any other has spurred the growth of
librarianship in this country and abroad.
.X
1       5       1
```

**Texto Original**

```
present studi histori dewey decim
classif first edit ddc publish
1876 eighteenth edit 1971 futur edit
continu appear need spite ddc
long healthi life howev full stori never
told biographi dewey
briefli describ system first
attempt provid detail histori work
spur growth
librarianship countri abroad
report analysi 6300 act use
104 technic librari unit kingdom
librari use one aspect wider pattern
inform use inform transfer librari
restrict use document take
account document use outsid librari still
less inform transfer oral person
person librari act channel
proport situat inform
transfer
take technic inform transfer whole
doubt proport
```

**Texto Preprocesado**

En el análisis final podemos encontrar palabras como “use”, un verbo que es considerado como stopword pero se repite muchas veces, que igualmente sirve para un analisis a profundidad, ya que con esto podemos obtener un promedio entre las palabras que se repiten

con mayor frecuencia y palabras que se mencionan una vez de todo el documento.

## V. CONCLUSIONES

Cumplimos entonces de manera satisfactoria los objetivos propuestos al inicio de la práctica tenemos en cuenta que podemos mejorar este programa agregando otras funciones, ya que es uno de los primeros pasos para un sistema más complejo como un buscador por ejemplo, igualmente puede contar con otros filtros, recordemos que nosotros podemos darle un significado a los datos que leamos, pero el programa y la computadora no pueden comprender como nosotros lo hacemos esa misma información, se basan en lo que nosotros agreguemos, por eso con más funciones podemos tener un análisis más preciso, pues el usuario final no somos nosotros sino otra persona que puede o no saber como usarlo, ahí es donde están nuestras funciones como programadores.

## VI. BIBLIOGRAFÍA


1. https://unipython.com/como-limpiar-el-texto-manualmente-usando-nltk/
2. https://likegeeks.com/es/tutorial-de-nlp-con-python-nltk/
3. https://www.google.com/amp/s/www.geeksforgeeks.org/removing-stop-words-nltk-python/amp/
4. https://ichi.pro/es/detenga-las-palabras-vacias-usando-diferentes-bibliotecas-de-python-281092180678227
5. https://gist.github.com/sebleier/554280
6. https://pythonprogramming.net/stemming-nltk-tutorial/
7. https://medium.com/qu4nt/reducir-el-n%C3%BAmero-de-palabras-de-un-texto-lematizaci%C3%B3n-y-radicalizaci%C3%B3n-stemming-con-python-965bfd0c69fa
8. https://medium.com/@roquelopez/algoritmo-de-porter-para-el-espa%C3%B1ol-en-java-dd44ea7b0a10