Trabajo Práctico

Análisis de Texto

Fecha de Entrega: 22/04/2021

Iván Costa

costaivan34@gmail.com

1. Escriba un programa que realice análisis léxico sobre la colección RI-tknz-data. El programa debe recibir como parámetros el directorio donde se encuentran los documentos y un argumento que indica si se deben eliminar las palabras vacías (y en tal caso, el nombre del archivo que las contiene). Defina, además, una longitud mínima y máxima para los términos. Como salida, el programa debe generar:

a) Un archivo (terminos.txt) con la lista de términos a indexar (ordenado), su frecuencia en la colección y su DF (Document Frequency).

b) Un segundo archivo (estadisticas.txt) con los siguientes datos:

1) Cantidad de documentos procesados

2) Cantidad de tokens y términos extraídos

3) Promedio de tokens y términos de los documento

4) Largo promedio de un término

5) Cantidad de tokens y términos del documento más corto y del más largo

6) Cantidad de términos que aparecen sólo 1 vez en la colección

c) Un tercer archivo (frecuencias.txt) con:

1) La lista de los 10 términos más frecuentes y su CF (Collection Frequency)

2) La lista de los 10 términos menos frecuentes y su CF.

El programa debe utilizar llamadas a las siguientes funciones:

a) lista tokens = tokenizar(string)2

b) lista terminos = sacar palabras vacias(lista tokens, lista vacias)

El código se encuentra en la carpeta /src/1. Para ejecutar el programa se debe ejecutar el siguiente comando:

python Tokenaizer.py [directorio_corpus] [Min_long_term] [Max_long_term] [directorio_palabras_vacías]

El primer parámetro es el directorio del corpus, el segundo es la longitud mínima de los términos, el tercero es la longitud máxima de los términos y el cuarto el directorio del archivo donde se encuentran las palabras vacías. Los resultados se guardan en la misma carpeta**,** con los nombres de archivos solicitados. Los últimos tres son opcionales (min=0,max=0,vacias=" ").

2. Tomando como base el programa anterior, escriba un segundo Tokenizer que implemente los criterios del artículo de Grefenstette y Tapanainen para definir qué es una “palabra” (o término) y cómo tratar números y signos de puntuación. Además, extraiga en listas separadas utilizando en cada caso una función específica.

a) Abreviaturas tal cual están escritas (por ejemplo, Dr., Lic., S.A., NASA, etc.)

b) Direcciones de correo electrónico y URLs

c) Números (por ejemplo, cantidades, teléfonos)

d) Nombres propios (por ejemplo, Villa Carlos Paz, Manuel Belgrano, etc.) y los trate como un único token.

Genere y almacene la misma información que en caso anterior.

El código se encuentra en la carpeta /src/2. Para ejecutar el programa se debe ejecutar el siguiente comando:

python TokenaizerV2.py [directorio_corpus] [Min_long_term] [Max_long_term] [directorio_palabras_vacías]

El primer parámetro es el directorio del corpus, el segundo es la longitud mínima de los términos, el tercero es la longitud máxima de los términos y el cuarto el directorio del archivo donde se encuentran las palabras vacías. Los resultados se guardan en la misma carpeta**,** con los nombres de archivos solicitados. Los últimos tres son opcionales (min=0,max=0,vacias=” “).

3. Repita el procesamiento del ejercicio 1 utilizando la colección RI-tknz-qm. Verifique los resultados e indique las reglas que debería modificar para que el tokenizer responda al dominio del problema.

Repitiendo el procesamiento con esta nueva colección, los resultados en general responden al dominio del problema. Esta colección posee textos sobre artículos referidos a procesos químicos. En casi todos los casos, los términos obtenidos mantienen la forma esperada por ejemplo con temperaturas, fracciones, formulas químicas, moléculas, porcentajes, etc. De igual forma seria interesante plantear expresiones regulares o funciones más específicas para identificar estos términos.

4. A partir del programa del ejercicio 1, incluya un proceso de stemming5. Luego de modificar su programa, corra nuevamente el proceso del ejercicio 1 y analice los cambios en la colección. ¿Qué implica este resultado? Busque ejemplos de pares de términos que tienen la misma raíz pero que el stemmer los trató diferente y términos que son diferentes y se los trató igual.

Se utiliza la librería nltk, con el stemmer Snowball, en su versión para el idioma español. El resultado de realizar este proceso fue que se redujo la lista de términos.

Los ejemplos de pares de términos que tienen la misma raíz pero el stemmer los trató de forma diferente son:

- computadora - computación
- escuela - escolar

Los ejemplos de pares de términos que son diferentes y se los trató igual son:

- parada - para
- palo - pala

El código se encuentra en la carpeta src/4/. Para ejecutar el programa se debe ejecutar el siguiente comando:

python TokenaizerV4.py [directorio_corpus] [Min_long_term] [Max_long_term] [directorio_palabras_vacías]

El primer parámetro es el directorio del corpus, el segundo es la longitud mínima de los términos, el tercero es la longitud máxima de los términos y el cuarto el directorio del archivo donde se encuentran las palabras vacías. Los resultados se guardan en la misma carpeta**,** con los nombres de archivos solicitados. Los últimos tres son opcionales (min=0,max=0,vacias=" ").

5. Sobre la colección CISI6, ejecute los stemmers de Porter y Lancaster provistos en el módulo nltk.stem. Compare: cantidad de tokens resultantes, resultado 1 a 1 y tiempo de ejecución para toda la colección. ¿Qué conclusiones puede obtener de la ejecución de uno y otro?

Las conclusiones que pude obtener en la utilización de los stemmers, está relacionado a la cantidad de términos únicos encontrados, en el caso del stemmer Lancaster resultaron una menor cantidad de términos únicos y además el promedio de longitud de términos resulto menor en comparación del stemmer Porter.

El código se encuentra en la carpeta /src/5. Para ejecutar el programa se debe ejecutar el siguiente comando:

python Tokenaizer.py [directorio_corpus] [Min_long_term] [Max_long_term] [directorio_palabras_vacías] [stemmer_a_utilizar]

El primer parámetro es el directorio del corpus, el segundo es la longitud mínima de los términos, el tercero es la longitud máxima de los términos y el cuarto el directorio del archivo donde se encuentran las palabras vacías. Los resultados se guardan en la misma carpeta, con los nombres de archivos solicitados. El último parámetro puede ser "porter" o "lancaster" dependiendo del stemmer a utilizar.

6. Escriba un programa que realice la identificación del lenguaje de un texto a partir de un conjunto de entrenamiento. Pruebe dos métodos sencillos:

a) Uno basado en la distribución de la frecuencia de las letras.

b) El segundo, basado en calcular la probabilidad de que una letra x preceda a una y (calcule una matriz de probabilidades con todas las combinaciones).

Compare los resultados contra el módulo Python langdetect8 y la solución provista.

Propiedades del Texto

7. En este ejercicio se propone verificar la predicción de ley de Zipf. Para ello, descargue desde Project Gutenberg el texto del Quijote de Cervantes10 y escriba un programa que extraiga los términos y calcule sus frecuencias. Calcule la curva de ajuste utilizando la función Polyfit del módulo NymPy. Con los datos crudos y los estimados grafique en la notebook ambas distribuciones (haga 2 gráficos, uno en escala lineal y otro en log-log). ¿Cómo se comporta la predicción? ¿Qué conclusiones puede obtener?

Puede observarse en los gráficos, que el documento dado cumple con la ley de Zipf. Además, se observa que el modelo se ajusta mejor sin los extremos del eje Ranking de Términos.

La solución se encuentra en el archivo **src/7y8/ejercicio7y8.ipynb**

8. Usando los datos del ejercicios anterior y de acuerdo a la ley de Zipf, calcule la proporción del total de términos para aquellos que tienen frecuencia f = f100; 1000; 10000g. Verifique respecto de los valores reales. ¿Qué conclusión puede obtener?

Conclusión: Al comparar los valores predecidos con los reales, puede decirse que las predicciones son aproximadas.

La solución se encuentra en el archivo **src/7y8/ejercicio7y8.ipynb**

9. Codifique un script que reciba como parámetro el nombre de un archivo de texto, tokenize y calcule los pares (#términos totales procesados, #términos ´únicos). Verifique en qué medida satisface la ley de Heaps. Grafique en la notebook los ajustes variando los parámetros de la expresión. Puede inicialmente probar con los archivos de los puntos 5 y 7.

La solución se encuentra en el archivo **src/9/ejercicio9.ipynb**