CAIM Lab, Session 1: ElasticSearch and Zipf's and Heaps' laws

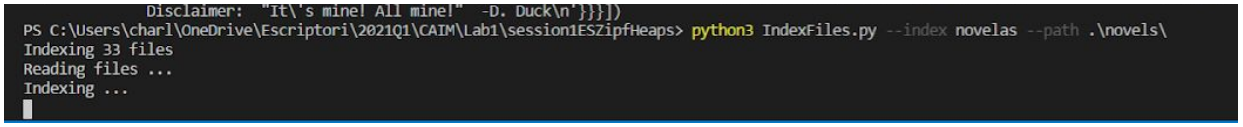
Moreno Oya, Daniel
Cidraque Sala, Carles

2020-2021Q1

Índex

Primero de todo lo que hacemos es indexar cada uno de los corpus proporcionados para este primer laboratorio.
Insertamos todos los documentos de **20_newsgroups** al index *news*:

Insertamos todos los documentos de **novels** al index *novelas*:



Insertamos todos los documentos de arxiv_abs al index *abstracts*:

Una vez hecha la indexación de los tres corpus pasamos a fijarnos en la distribución de palabras que tienen estos documentos que hay en los diferentes 3 corpus que hemos descargado.

En particular vamos a mirar si se siguen las leyes de **Zipf** y **Heap**.

Ley de Zipf

Primero empezaremos con la ley de **Zipf**.

Lo que hacemos es utilizar el script proporcionado en este primer laboratorio llamado CountWords.py que lo que hace es leer un índice y escribir por la salida estandard todas las palabras que contiene ese índice con el número de veces que aparece cada una de ellas.

Por lo tanto, lo que haremos primero es ejecutar este script mencionado sobre el corpus de novelas, redirigir la salida hacia un fichero de texto plano. Esta salida la ordenamos alfabéticamente mediante la introducción del flag --alpha en la ejecución del script.

Una vez obtuvimos la salida ordenada alfabéticamente nos dimos cuenta que era mucho más fàcil si no lo estaba, es decir, si estaba ordenada por frecuencia (número de ocurrencias de los términos en el texto) para así obtener de una forma más ràpida i fàcil la gràfica frecuencia-rango directamente del fichero obtenido de la ejecución del script CountWords.py.

Una vez obtenida este output mencionado, procedimos a eliminar aquellos términos que sean poco útiles, o bien posibles errores que pueda haber en determinados textos de un corpus en concreto (eliminando términos que contengan “.”, “_”, “:”, ...), así solo dejando únicamente esas palabras que podríamos encontrar en un diccionario por ejemplo.

Una vez hecha esta limpieza estamos listos para analizar los datos.

Para la ley de **Zipf** tenemos que mirar si la distribución de frecuencia-rango se asemeja y sigue la distribución de la siguiente función: **f = c / (rank + b)^a**

Para hacerlo hemos seleccionado los primeros 500 términos más frecuentes obtenidos a partir del output generado por el script CountWords.py sobre el índice creado novelas.
Vemos que al dibujar la distribución de frecuencia-rango obtenemos la curva decreciente ya esperada. Vemos que muestra grandes frecuencias para los términos con primeros rangos, pero en seguida vemos como disminuye esta frecuencia para rangos más elevados.
Podemos ver que el término de rango = 1 tiene una frecuencia = 206546 mientras que el término último mostrado de rango = 500 tiene una frecuencia = 612.

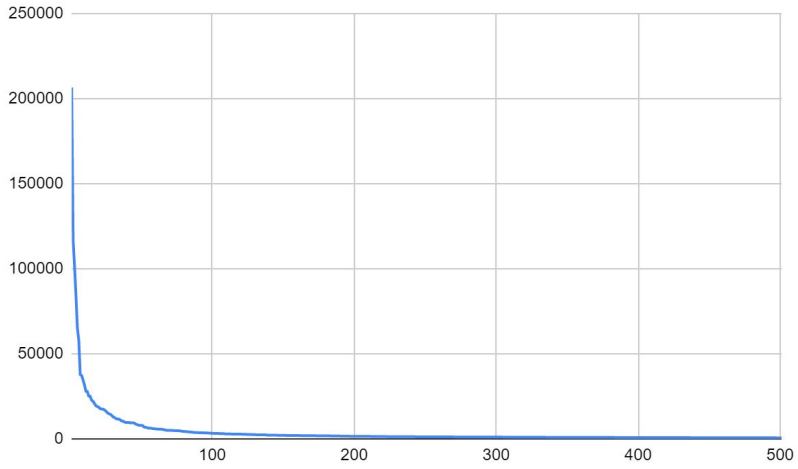


Fig 1. gráfica freq- rango real (de las 500 términos con mayor frecuencia)

Tenemos que averiguar cuales son los parámetros “a”,”b” y “c” de la fórmula de Zipf y mirar si el resultado que obtenemos con esa parametrización se asemeja a la distribución real que vemos en los datos del corpus.

Para ello hemos hecho uso de la librería *scipy* de python que nos ha permitido hacer uso del método *curve_fit* que nos ajusta a valores óptimos los parámetros.

Para empezar decidimos dejar el ajuste de todos los parámetros de la fórmula Zipf a los valores que curve_fit decidiera óptimos, pero no obtubimos resultados satisfactorios, ya que la gráfica de frecuencia-rango generada no se ajustaba a la obtenida anteriormente (fig1) mediante el programa Excel a partir de los 500 términos más frecuentes.

Decidimos fijar el valor de a = 1 y los parámetros “b” y “c” que los ajustara curve_fit.
A partir de aquí obtenemos que “b” y “c” son igual a 0.7739336760651987
360984.199149414 respectivamente.
Procedimos a partir de estos valores a generar el plot de tal forma que se pudiese comparar con el generado en la *fig1*, hasta el rango 500 y a partir de la frecuencia más alta obtenida del término de rango 1: plt.axis([0,500, 0, 206546]);. Así obteniendo la gráfica mostrada en la *fig2* donde podemos ver que se asemeja mucho al obtenido en la *fig1*.

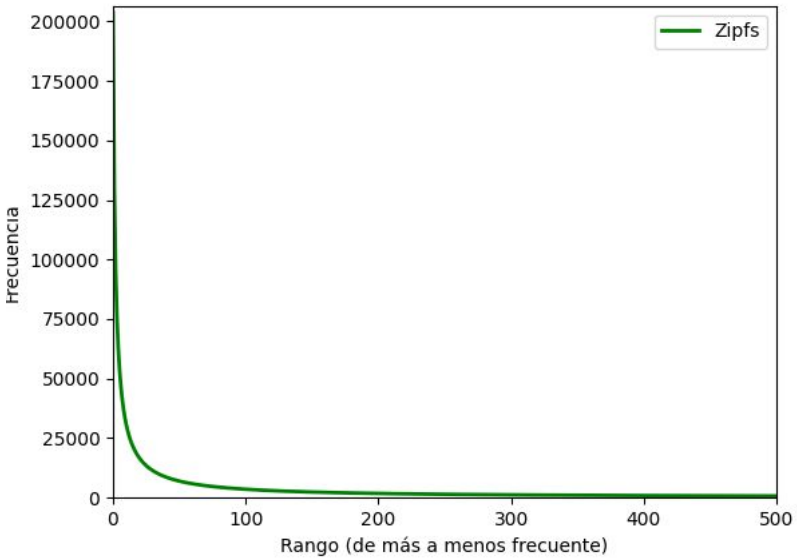


fig2. gráfica frecuencia-rango Zipf (de las 500 términos con mayor frecuencia)

También generamos un grafo aplicando log (*fig3*) obtenemos que se ajusta también bastante bien a la de Zipf, aunque vemos como empiezan a diferir a medida que llegamos a valores de rango superiores, donde la ocurrencia de cada uno de los términos analizados es más baja y es dónde se acumulan la mayoría de palabras de los diferentes documentos.

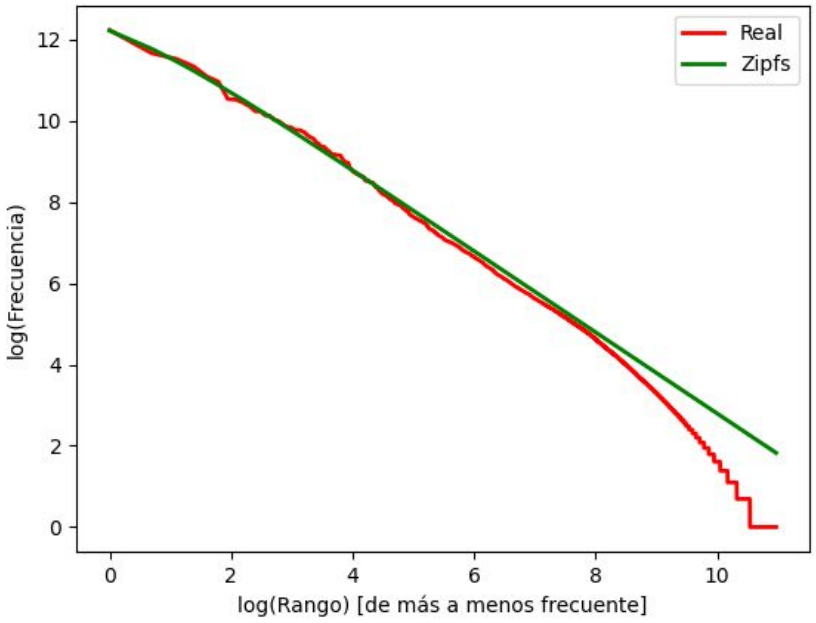


fig3. gráfica log(frecuencia)-log(rango) de la muestra real i a partir de Zipf (de las 500 términos con mayor frecuencia)

Ley de Heap

Ahora pasamos a analizar la ley de **Heap**.
Teníamos que mirar si el numero de términos, de palabras diferentes (d) que aparecen en un texto que tiene “N” palabras contiene aproximadamente $k*N^B$ términos diferentes.

$$d = k*N^B$$

Para ello, solo hemos analizado en este caso el corpus de novelas tal y como nos facilitaba el enunciado del laboratorio.

Primero lo que hemos hecho es crear índices diferentes que contengan diferentes números de novelas, es decir, diferente cantidad de texto. Para ello hemos creado a partir del corpus inicial de novelas(indexado como “novelas” al inicio del laboratorio) otros 3 índices más cada uno con la mitad aproximada de texto que el anterior. (tamaño *novelas* = 17.4 MB, tamaño *novelss* = 8.98MB, tamaño *novelsss* = 4.62MB tamaño *novelssss* = 2.18MB).

Después tuvimos que contar cuántas palabras diferentes contenían cada uno de los índices, esto lo hicimos con el script CountWords.py dónde al final de su output nos indica el número total de términos diferentes encontrados.
En concreto, el índice *novelas* contiene 61825 palabras distintas, el *novelss* 39047, *novelsss 25805* y *novelssss* 18165.
También hemos limpiado estas salidas de los términos que no consideramos palabras tal y como hemos hecho en el análisis de Zipf. Por lo tanto, tuvimos que hacer un recuento de palabras adicional, ya que el ofrecido por CountWords.py solo nos ofrecía una aproximación.
Para contar el número total de términos N que tiene cada uno de los índices lo calculamos haciendo la suma de cada uno del número de ocurrencias de cada término del fichero generado por CountWords.py por cada índice.

Obtenemos:

```
PS C:\Users\charl\OneDrive\Escriptori\2021Q1\CAIM\Lab1\session1ESZipfHeaps> python3 grafHeap.py
TXTnovelas.txt Numero total de palabras =  3138789.0 Palabras diferentes =  58114
TXTnovelassAUX.txt Numero total de palabras =  1615215.0 Palabras diferentes =  37956
TXTnovelasssAUX.txt Numero total de palabras =  839325.0 Palabras diferentes =  25609
TXTnovelassssAUX.txt Numero total de palabras =  397783.0 Palabras diferentes =  18152
```

fig4. output generado para ver número total de palabras y número total de palabras diferentes de cada índice ordenados por tamaño.

Una vez obtenidos para cada índice el número de términos totales y el número de términos diferentes, procedemos a utilizar el método también utilizado en el estudio de Zipf, *curve_fit*. Éste a partir de la fórmula de Heaps y el parámetro N y las “soluciones reales” D ajusta los parámetros *k* i *B*. Obtenemos una *k* i una *B* de 7.8452166693039915 0.595056559300275 respectivamente. A partir de aquí ya podemos generar los plots correspondientes y ver cómo se ajusta las soluciones obtenidas a partir de Heaps y las reales. Vemos que se ajusta muy bien los resultados predichos por Heap a los reales.

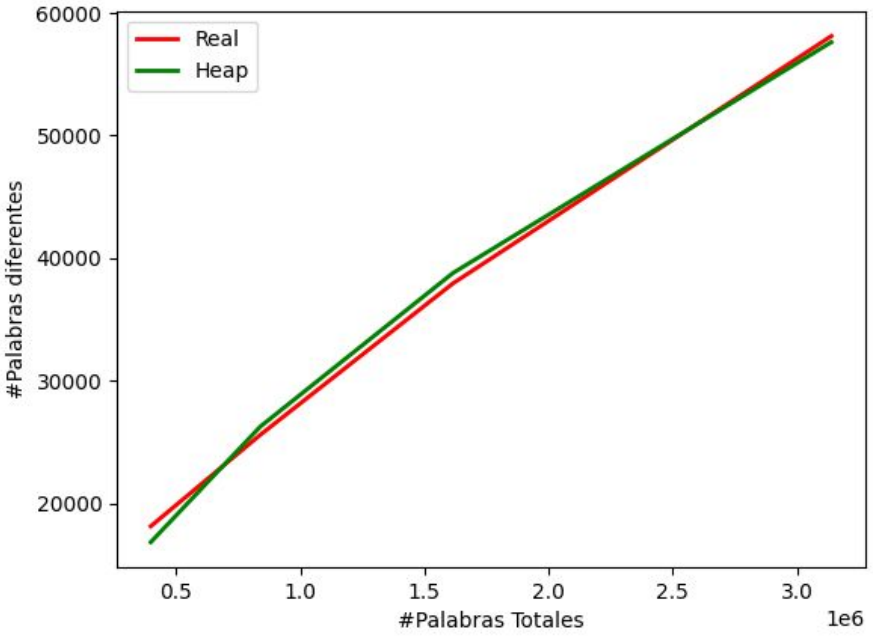


fig5. Gráfica real y Heap del número de palabras diferentes respecto al número total de palabras.

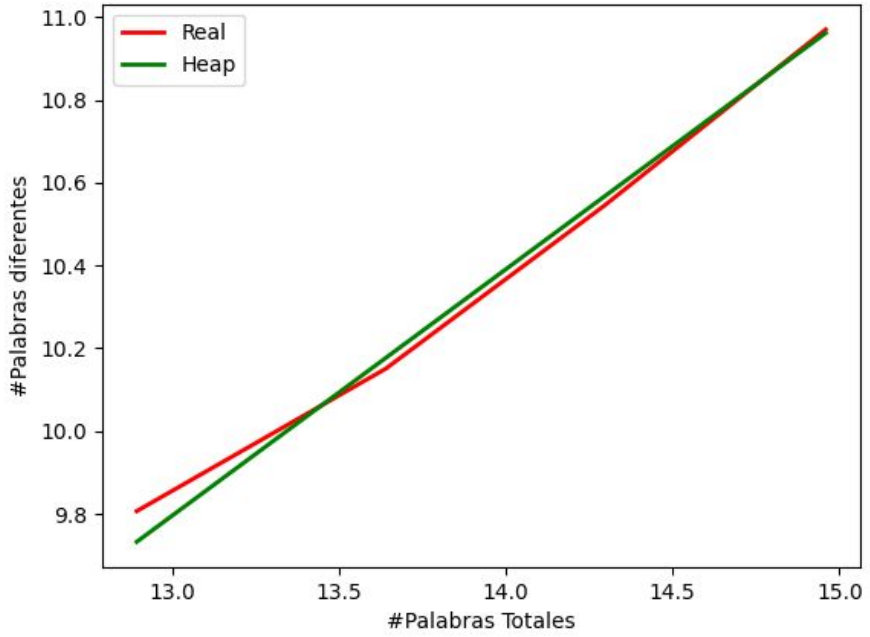


fig6. Gráfica real y Heap del log(número de palabras diferentes) respecto al log(número total de palabras).