TP1: Análisis de Texto

Alumno: Gallozo, Luis Angel 156308

1. Escriba un programa que realice analisis lexico sobre la coleccion RI-tknz-data. El programa debe recibir como parametros el directorio donde se encuentran los documentos y un argumento que indica si se deben eliminar las palabras vacas (y en tal caso, el nombre del archivo que las contiene). Dena, ademas, una longitud mnima y maxima para los terminos. Como salida, el programa debe generar:
a) Un archivo (terminos.txt) con la lista de terminos a indexar (ordenado), su frecuencia en la coleccion y su DF (Document Frequency). Formato de salida: < termino > [ESP] < CF > [ESP] < DF >.
Ejemplo:
casa 238 3
perro 644 6
...
zorro 12 1
b) Un segundo archivo (estadisticas.txt) con los siguientes datos (un punto por lnea y separados por espacio cuando sean mas de un valor) :
1) Cantidad de documentos procesados
2) Cantidad de tokens y terminos extrados
3) Promedio de tokens y terminos de los documentos
4) Largo promedio de un termino
5) Cantidad de tokens y terminos del documento mas corto y del mas largo1
6) Cantidad de terminos que aparecen solo 1 vez en la coleccion
c) Un tercer archivo (frecuencias.txt, con un termino y su CF por lnea) con:
1) La lista de los 10 terminos mas frecuentes y su CF (Collection Frequency)
2) La lista de los 10 terminos menos frecuentes y su CF.

2. Tomando como base el programa anterior, escriba un segundo Tokenizer que implemente los criterios del artculo de Grefenstette y Tapanainen para denir que es una \palabra" (o termino) y como tratar numeros y signos de puntuacion. Ademas, extraiga en listas separadas utilizando en cada caso una funcion especca.
a) Abreviaturas tal cual estan escritas (por ejemplo, Dr., Lic., S.A., etc.)2
b) Direcciones de correo electronico y URLs.
c) Numeros (por ejemplo, cantidades, telefonos).
d) Nombres propios (por ejemplo, Villa Carlos Paz, Manuel Belgrano, etc.) y los trate como un unico token.
Genere y almacene la misma informacion que en el caso anterior.

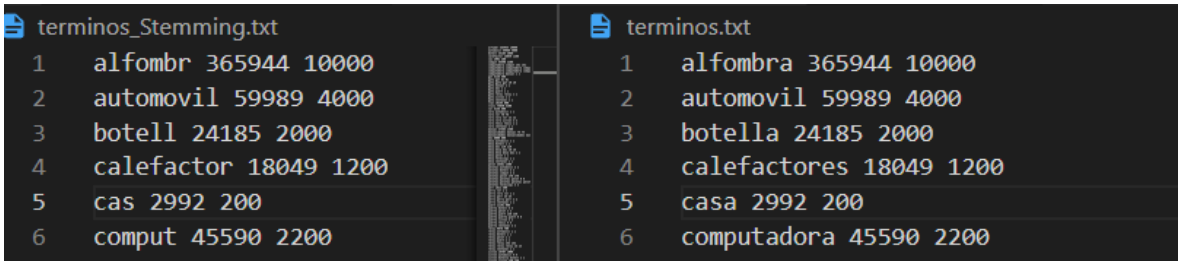
Los puntos 1 y 2 solo son código, las conclusiones de los demás puntos estarán en este documento.

Aclaraciones:

1. Se decidió un mínimo de 2 y máximo de 50 caracteres para los tokens.
2. Al extraer Nombre propios se decide dejar los tokens que lo conforman, ya que no daban bien las frecuencias según los datos de prueba que nos dieron.
3. Al extraer las URL, se decide quitar toda la url de la lista de tokens.
4. Al extraer Cantidad, se tienen en cuenta todos los números reales enteros y floats (negativos y positivos), también los números de teléfono.
5. Los top 10 no están ordenados por orden alfabético, ya que se van almacenando según son extraídos si son menores o mayores a los almacenados se adhieren a las listas.

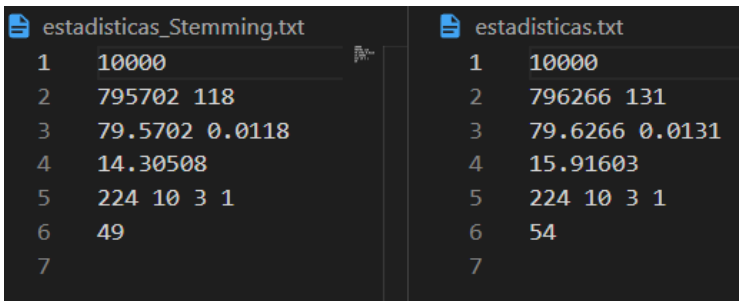
3. A partir del programa del ejercicio 1, incluya un proceso de stemming. Luego de modificar su programa, corra nuevamente el proceso del ejercicio 1 y analice los cambios en la colección. ¿Qué implica este resultado?
Busque ejemplos de pares de términos que tienen la misma raÍz pero que el stemmer los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

Como primera observación todos los términos almacenados sufrieron el cambio a sus raíces morfológicas:



En el apartado de las estadísticas, sufren grandes cambios, como por ejemplo:

- Línea 2: La cantidad de tokens se redujo a 795702 y los términos a 118
- Línea 4: El largo promedio de un término se reduce
- Línea 6: La cantidad de términos con frecuencia 1 también se reducen.



4. Sobre la colección CISI, ejecute los stemmers de Porter y Lancaster provistos en el módulo nltk.stem.
Compare: cantidad de tokens únicos resultantes, resultado 1 a 1 y tiempo de ejecución para toda la colección.
¿Qué conclusiones puede obtener de la ejecución de uno y otro?
En los resultados de los términos más frecuentes(10 primeros) y menos frecuentes (10 últimos), tenemos:

- No tienen los mismos términos en sus rankings.
- Las frecuencias en Lancaster son superiores, además de que la reducción de algunos términos es notable.

frecuencias_Stemming_Lancas...

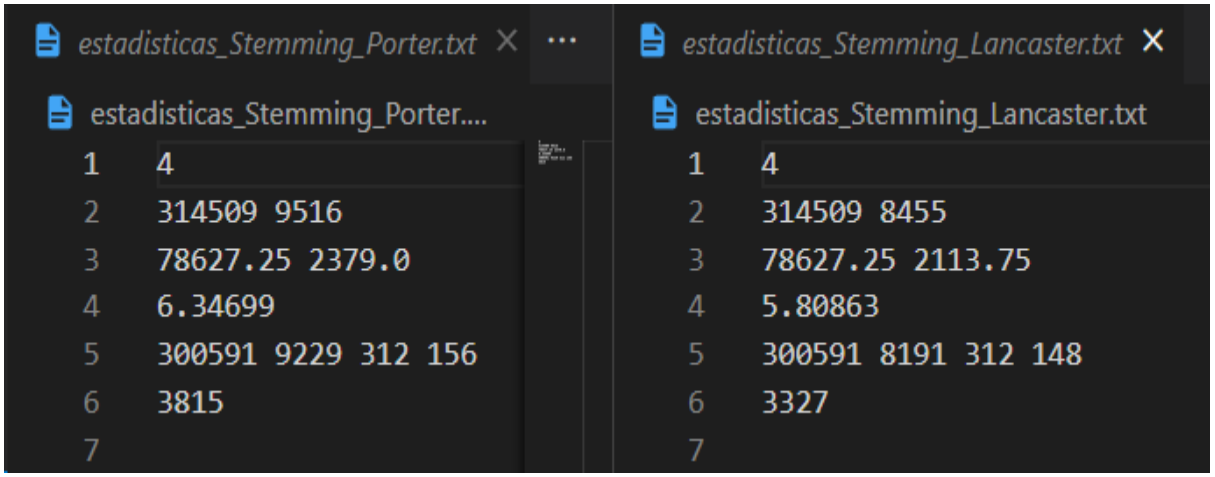
```
the 14176
and 6736
for 2530
libr 2235
ar 1845
inform 1813
us 1775
that 1619
system 1473
zyabrev 1
zyabrev 1
zvezhinski 1
zoolog 1
zip 1
zimmerm 1
zilbermint 1
zhurn 1
zholkovski 1
zhdanova 1
zel 1
```

frecuencias_Stemming_Porter.txt

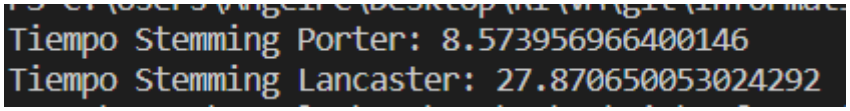
```
the 13924
and 6722
for 2520
librari 1915
are 1800
inform 1796
that 1619
thi 1467
system 1421
zyabrev 1
zyabrev 1
zvezhinskii 1
zoolog 1
zipper 1
zipfian 1
zimmerman 1
zilbermint 1
zhurnal 1
zholkovskii 1
zhdanova 1
```

En el apartado de las estadísticas, sufren grandes cambios, como por ejemplo:

- Línea 2: La cantidad de tokens y términos extraídos. **Lancaster extrajo menos términos.**
- Línea 3: Promedio de tokens y términos de los documentos. **Lancaster extrajo menos términos en promedio.**
- Línea 4: El largo promedio de un término se reduce. **Lancaster tuvo menos longitud promedio de términos.**
- Línea 5: Cantidad de tokens y términos del documento más corto y del más largo. **Lancaster obtuvo tamaño de documentos menores a Porter, además se refleja en la cantidad de tokens y términos que extrajo.**
- Línea 6: La cantidad de términos con frecuencia 1. **Lancaster pudo reducir la cantidad de términos con frecuencia.**



En tiempos de ejecución tenemos que el stemmer de Lancaster tarda más de una tercera parte que el de Porter.



Al observar ambos stemmers se denota como en cuestiones de rigurosidad para reducir cada término término afectando en gran medida al tiempo total de ejecución. Al momento de elegir cual utilizar debemos plantearnos el objetivo del sistema, además de elegir que priorizar, el uso eficiente de recursos o el tiempo de respuesta.

5. 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 langdetect y la solución provista.

Comparación entre el punto a) y b):
Se diseñó un script(“verificar_resultados_lang.py”) que puede comparar las diferencias entre 2 archivos resultado. Como se ve en la imagen el a) y b) tienen muchas oraciones con calificación distinta, haremos la prueba con el archivo solution.

```
Oracion: [1] Lang_1: French Lang_2: Italian
Oracion: [6] Lang_1: Italian Lang_2: English
Oracion: [13] Lang_1: Italian Lang_2: English
Oracion: [21] Lang_1: French Lang_2: Italian
Oracion: [22] Lang_1: French Lang_2: Italian
Oracion: [28] Lang_1: French Lang_2: English
Oracion: [38] Lang_1: French Lang_2: English
Oracion: [40] Lang_1: English Lang_2: Italian
Oracion: [44] Lang_1: French Lang_2: Italian
Oracion: [46] Lang_1: French Lang_2: Italian
Oracion: [51] Lang_1: Italian Lang_2: English
Oracion: [65] Lang_1: English Lang_2: Italian
Oracion: [74] Lang_1: Italian Lang_2: English
Oracion: [85] Lang_1: French Lang_2: Italian
Oracion: [87] Lang_1: Italian Lang_2: French
Oracion: [93] Lang_1: Italian Lang_2: French
Oracion: [103] Lang_1: French Lang_2: English
Oracion: [118] Lang_1: French Lang_2: Italian
Oracion: [120] Lang_1: English Lang_2: Italian
Oracion: [126] Lang_1: French Lang_2: Italian
Oracion: [129] Lang_1: Italian Lang_2: French
Oracion: [140] Lang_1: Italian Lang_2: English
Oracion: [141] Lang_1: Italian Lang_2: French
Oracion: [169] Lang_1: English Lang_2: Italian
Oracion: [176] Lang_1: English Lang_2: French
Oracion: [206] Lang_1: French Lang_2: English
Oracion: [215] Lang_1: French Lang_2: English
Oracion: [226] Lang_1: French Lang_2: Italian
Oracion: [239] Lang_1: Italian Lang_2: English
Oracion: [246] Lang_1: English Lang_2: Italian
Oracion: [251] Lang_1: French Lang_2: English
Oracion: [258] Lang_1: French Lang_2: Italian
Oracion: [261] Lang_1: French Lang_2: Italian
Oracion: [265] Lang_1: French Lang_2: English
Oracion: [271] Lang_1: French Lang_2: English
Oracion: [272] Lang_1: Italian Lang_2: French
Oracion: [281] Lang_1: Italian Lang_2: English
Oracion: [297] Lang_1: French Lang_2: Italian
```

Comparación punto a) con el archivo solution:

```
Oracion: [1] Lang_1: French Lang_2: Italian
Oracion: [6] Lang_1: Italian Lang_2: English
Oracion: [13] Lang_1: Italian Lang_2: English
Oracion: [21] Lang_1: French Lang_2: Italian
Oracion: [28] Lang_1: French Lang_2: English
Oracion: [38] Lang_1: French Lang_2: English
Oracion: [40] Lang_1: English Lang_2: Italian
Oracion: [44] Lang_1: French Lang_2: Italian
Oracion: [46] Lang_1: French Lang_2: Italian
Oracion: [51] Lang_1: Italian Lang_2: English
Oracion: [65] Lang_1: English Lang_2: Italian
Oracion: [74] Lang_1: Italian Lang_2: English
Oracion: [85] Lang_1: French Lang_2: Italian
Oracion: [87] Lang_1: Italian Lang_2: English
Oracion: [93] Lang_1: Italian Lang_2: French
Oracion: [103] Lang_1: French Lang_2: English
Oracion: [113] Lang_1: French Lang_2: English
Oracion: [118] Lang_1: French Lang_2: Italian
Oracion: [120] Lang_1: English Lang_2: Italian
Oracion: [129] Lang_1: Italian Lang_2: French
Oracion: [140] Lang_1: Italian Lang_2: English
Oracion: [141] Lang_1: Italian Lang_2: French
Oracion: [169] Lang_1: English Lang_2: Italian
Oracion: [176] Lang_1: English Lang_2: French
Oracion: [201] Lang_1: Italian Lang_2: English
Oracion: [206] Lang_1: French Lang_2: English
Oracion: [215] Lang_1: French Lang_2: English
Oracion: [226] Lang_1: French Lang_2: Italian
Oracion: [239] Lang_1: Italian Lang_2: English
Oracion: [246] Lang_1: English Lang_2: Italian
Oracion: [251] Lang_1: French Lang_2: English
Oracion: [258] Lang_1: French Lang_2: Italian
Oracion: [261] Lang_1: French Lang_2: Italian
Oracion: [265] Lang_1: French Lang_2: English
Oracion: [271] Lang_1: French Lang_2: English
Oracion: [272] Lang_1: Italian Lang_2: French
Oracion: [281] Lang_1: Italian Lang_2: English
Oracion: [297] Lang_1: French Lang_2: Italian
```

Comparación punto b) con el archivo solution:

Como se ve, el punto b) fue más certero en la clasificación debido a un análisis más exhaustivo de pares de caracteres.

```
Oracion: [22] Lang_1: Italian Lang_2: French
Oracion: [87] Lang_1: French Lang_2: English
Oracion: [113] Lang_1: French Lang_2: English
Oracion: [126] Lang_1: Italian Lang_2: French
Oracion: [201] Lang_1: Italian Lang_2: English
```

Comparación con el módulo Python langdetect:

1. Comparación entre punto a) y langdetect:

```
Oracion: [1] Lang_1: French Lang_2: Italian
Oracion: [6] Lang_1: Italian Lang_2: English
Oracion: [13] Lang_1: Italian Lang_2: English
Oracion: [21] Lang_1: French Lang_2: Italian
Oracion: [28] Lang_1: French Lang_2: English
Oracion: [38] Lang_1: French Lang_2: English
Oracion: [40] Lang_1: English Lang_2: Italian
Oracion: [44] Lang_1: French Lang_2: Italian
Oracion: [46] Lang_1: French Lang_2: Italian
Oracion: [51] Lang_1: Italian Lang_2: English
Oracion: [65] Lang_1: English Lang_2: Italian
Oracion: [74] Lang_1: Italian Lang_2: English
Oracion: [85] Lang_1: French Lang_2: Italian
Oracion: [87] Lang_1: Italian Lang_2: Indonesian
Oracion: [93] Lang_1: Italian Lang_2: French
Oracion: [103] Lang_1: French Lang_2: English
Oracion: [113] Lang_1: French Lang_2: English
Oracion: [118] Lang_1: French Lang_2: Italian
Oracion: [120] Lang_1: English Lang_2: Italian
Oracion: [126] Lang_1: French Lang_2: Italian
Oracion: [129] Lang_1: Italian Lang_2: French
Oracion: [140] Lang_1: Italian Lang_2: English
Oracion: [141] Lang_1: Italian Lang_2: French
Oracion: [169] Lang_1: English Lang_2: Italian
Oracion: [176] Lang_1: English Lang_2: French
Oracion: [201] Lang_1: Italian Lang_2: Somali
Oracion: [206] Lang_1: French Lang_2: English
Oracion: [215] Lang_1: French Lang_2: English
Oracion: [226] Lang_1: French Lang_2: Italian
Oracion: [239] Lang_1: Italian Lang_2: English
Oracion: [246] Lang_1: English Lang_2: Italian
Oracion: [251] Lang_1: French Lang_2: English
Oracion: [258] Lang_1: French Lang_2: Italian
Oracion: [261] Lang_1: French Lang_2: Italian
Oracion: [265] Lang_1: French Lang_2: English
Oracion: [271] Lang_1: French Lang_2: English
Oracion: [272] Lang_1: Italian Lang_2: French
Oracion: [281] Lang_1: Italian Lang_2: English
Oracion: [297] Lang_1: French Lang_2: Italian
```

2. Comparación entre punto b) y langdetect:

```
Oracion: [22] Lang_1: Italian Lang_2: French
Oracion: [87] Lang_1: French Lang_2: Indonesian
Oracion: [113] Lang_1: French Lang_2: English
Oracion: [201] Lang_1: Italian Lang_2: Somali
```

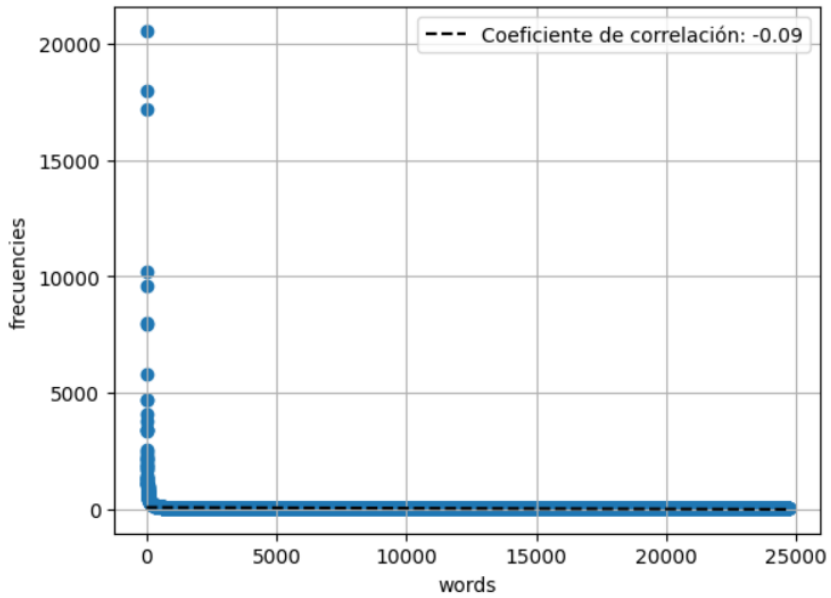
3. Comparación entre langdetect y archivo solution :

```
Oracion: [87] Lang_1: Indonesian Lang_2: English
Oracion: [126] Lang_1: Italian Lang_2: French
Oracion: [201] Lang_1: Somali Lang_2: English
```

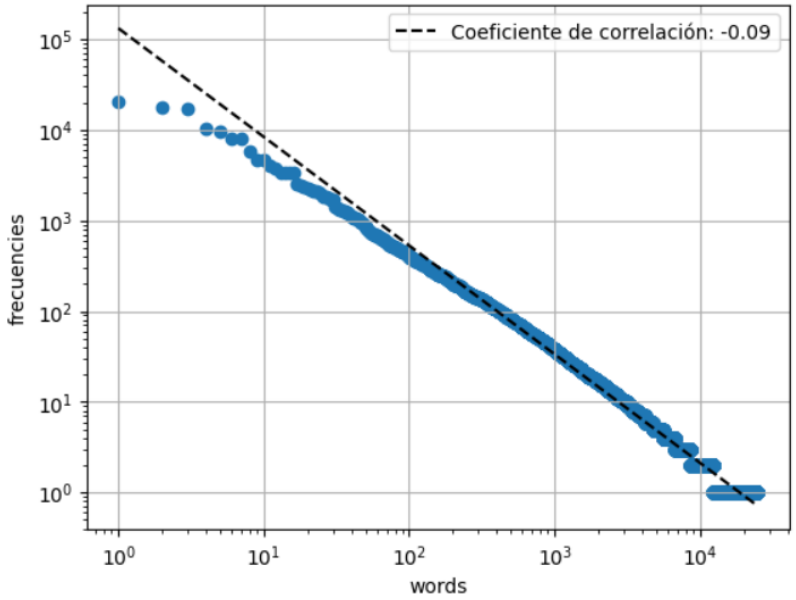
Como se observa en las comparaciones la mejora del punto b es notable, pero no tan buena como el módulo langdetect, aún así sorprende la aparición de nuevos idiomas, obviamente el módulo contiene mucha más robustez y mejor lógica para realizar las comparaciones.

6. En este ejercicio se propone verificar la predicción de ley de Zipf. Para ello, descargue desde Project Gutenberg el texto del Quijote de Cervantes y escriba un programa que extraiga los términos y calcule sus frecuencias (el programa debe generar la lista ordenada por frecuencia descendente). Calcule la curva de ajuste utilizando la función Polyfit del modulo NymPy9. Con los datos crudos y los estimados grafique en la notebook ambas distribuciones (haga 2 graficos, uno en escala lineal y otro en log-log). ¿Cómo se comporta la predicción? ¿Qué conclusiones puede obtener?

Escala Lineal:



Escala Logarítmica:

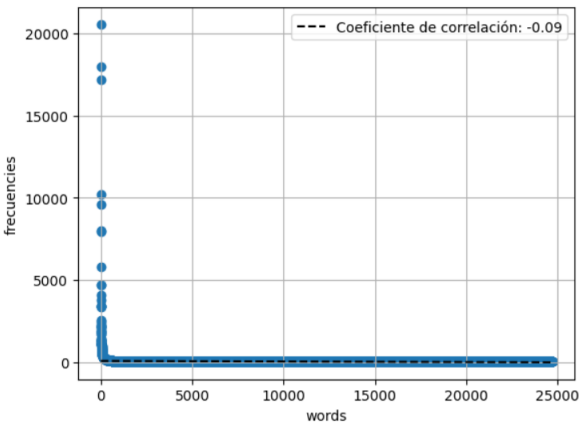


Como se puede ver en ambos gráficos existe un coeficiente de correlación de -0.09, indica que hay una relación lineal negativa entre la frecuencia y los términos, lo cual quiere decir que mientras más aumenten los términos aumenta la aparición de términos poco frecuentes. Sin embargo, el valor pendiente sugiere que esta relación es bastante débil en el texto analizado.

Por lo que se identifica que en el archivo se sigue la ley de Zypf. La distribución de frecuencia sigue esta relación inversa entre la frecuencia y el rango, en la que los términos más comunes tienen un rango bajo y las menos comunes tienen un rango alto. Así que lo que estaría pasando es que los términos más frecuentes en el archivo son las palabras vacías(stopwords). Una ayuda a esto sería determinar los términos más significativos y ponderarlos más que las stopwords.

7.Usando los datos del ejercicio anterior y de acuerdo a la ley de Zipf, calcule la proporción del total de términos para aquellos que tienen frecuencia f = {100; 1000; 10000}. Verifique respecto de los valores reales.¿Qué conclusión puede obtener?

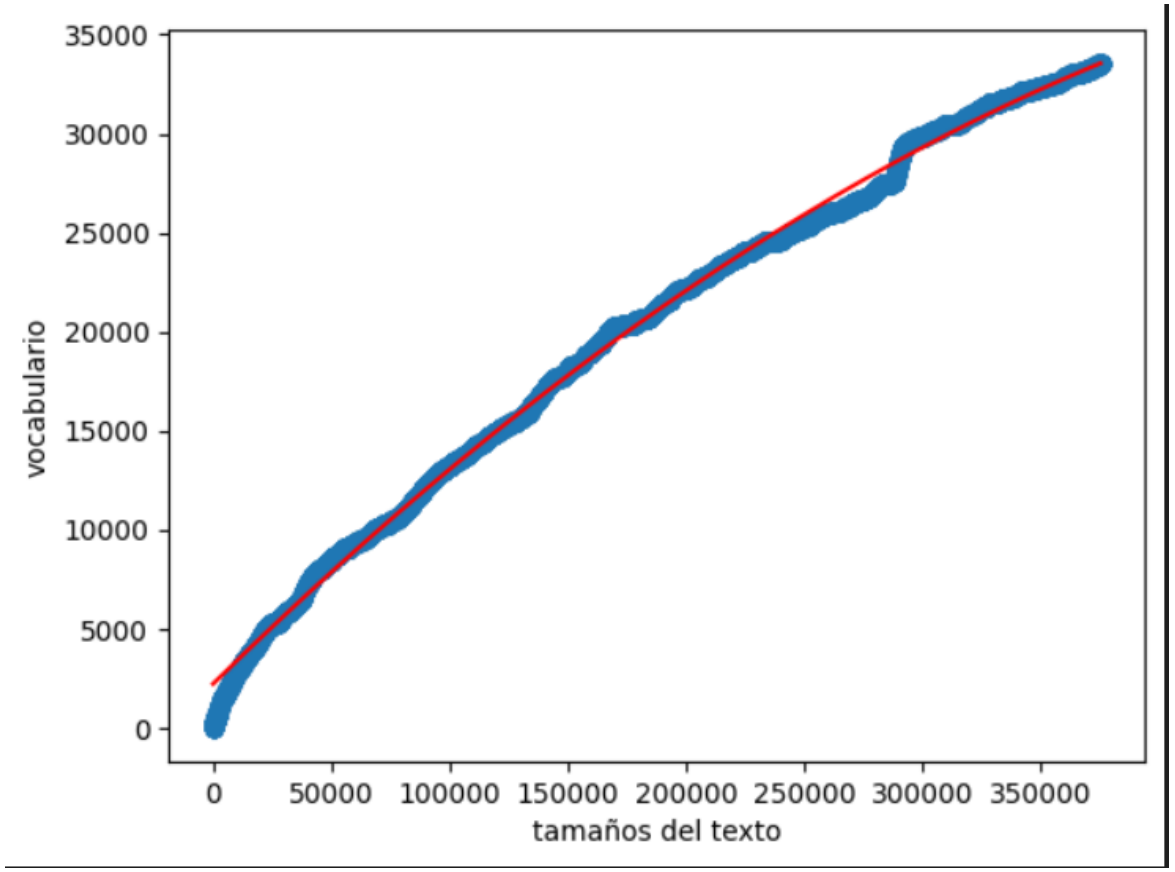
```
Escala Normal:

Proporcion frec 100: 56.452666891317094

Proporcion frec 1000: 53.44747827730361

Proporcion frec 10000: 23.395592137168805


Escala Logaritmica:

Proporcion frec 100: 529.9365189599802

Proporcion frec 1000: 33.55701570197457

Proporcion frec 10000: 2.1249211226894293
```



Como se ve en los resultados, coinciden con la ley de Zypf, ya que los términos más frecuentes tienen un porcentaje bajo, mientras que los menos frecuentes tienen un porcentaje alto.

8. Codifique un script que reciba como parámetro el nombre de un archivo de texto, tokenize y calcule y escriba a un archivo 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 anteriores.

Se realizó un script(“punto_8_heaps.py”) que analiza todo un directorio y por cada archivo procesado escribe un registro de lo pedido. Se procesaron los archivos txt de “RI-tknz-data”, dando los siguientes resultados:



Como se aprecia en el gráfico se cumple la ley de Heaps, ya que a medida que se iban procesando documentos en el script, se descubrirán nuevos términos únicos (aumentando el vocabulario). También se ve que cada que crece la cantidad de documentos procesados, la adición de términos únicos en el vocabulario serán menos, ya que hay probabilidad de que los documentos compartan términos entre sí.