Trabajo Práctico
Análisis de Texto

Fecha entrega: 02/04/2025

Para la entrega del TP resuelto arme un único archivo (.pdf) y envíelo a través del formulario correspondiente[1], (el cual se encontrará habilitado hasta la fecha de entrega establecida).

1) Utilizando la colección para debugging[2] provista por el equipo docente escriba un programa qe realice el análisis léxico sobre la misma y extraiga la siguiente información:

   - Lista de términos y su DF
   - Cantidad de *tokens*
   - Cantidad de términos
   - Cantidad de documentos procesados

   Compare los resultados obtenidos por su programa con los que figuran en el archivo *collection_data.json* que contiene los metadatos de la colección.

2) 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:

   - Un archivo (`terminos.txt`) con la lista de términos a indexar (ordenado), su frecuencia en la colección y su DF (*Document Frequency*).
     Formato de salida: `$<termino> [ESP] <CF> [ESP] <DF>$`.
     Ejemplo:
     ```
     casa 238 3
     perro 644 6
     ...
     zorro 12 1
     ```

   - Un segundo archivo (`estadisticas.txt`) con los siguientes datos (un ítem por línea y separados por espacio cuando sean más de un valor):

[1] Link formulario entrega: https://forms.gle/1M39xwK2ckpks3D77
[2] https://drive.google.com/file/d/1-Oz6ffJoKsTZM5S4gaPx6nzrxfVOVOG_/view?usp=sharing

- Cantidad de documentos procesados.
- Cantidad de tokens y términos extraídos.
- Promedio de tokens y términos de los documentos.
- Largo promedio de un término.
- Cantidad de *tokens* y términos del documento más corto y del más largo[3].
- Cantidad de términos que aparecen sólo 1 vez en la colección.

- Un tercer archivo (`frecuencias.txt`), con:
  - La lista de los 10 términos más frecuentes y su CF (Collection Frequency). Un término por línea.
  - La lista de los 10 términos menos frecuentes y su CF. Un término por línea.

3) 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. En este caso su tokenizer deberá extraer y tratar como un único término:

   - Abreviaturas tal cual están escritas (por ejemplo, Dr., Lic., S.A., etc.)[4]
   - Direcciones de correo electrónico y URLs.
   - Números (por ejemplo, cantidades, teléfonos).
   - Nombres propios (por ejemplo, Villa Carlos Paz, Manuel Belgrano, etc.)

   Utilice la colección para *debugging*[5] de expresiones regulares provista por el equipo docente para extraer y comparar la salida de su programa con los metadatos de la colección tal como lo realizó en el punto 1.

   Por último, extraiga y almacene la misma información que en el punto 2 sobre la colección **RI-tknz-data** utilizando su nuevo *tokenizer*.

4) A partir del programa del ejercicio 1, incluya un proceso de *stemming*[6]. Luego de modificar su programa, corra nuevamente el proceso del ejercicio 2 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.

[3] Mida la longitud del documento en cantidad de *tokens*.
[4] Discuta cómo resolvería la extracción de abreviaturas como "NASA".
[5] https://drive.google.com/file/d/1mP4t7HlDF66joGV9CPok8ZZxC1N-J5LT/view
[6] Puede usar la librería NLTK (Natural Language Toolkit). Revise qué algoritmos soporta (español e inglés).

5) Sobre la colección Vaswani[7], ejecute los *stemmers* 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?

6) Escriba un programa que realice la identificación del lenguaje de un texto a partir de un conjunto de entrenamiento[8]. Pruebe dos métodos sencillos:
   - Uno basado en la distribución de la frecuencia de las letras.
   - El segundo, basado en calcular la probabilidad de que una letra *x* preceda a otra letra *y* (calcule una matriz de probabilidades con todas las combinaciones).

   Compare los resultados contra el módulo Python `langdetect`[9] y la solución provista.

[7] https://github.com/tolosoft-academia/RI_2025/tree/main/data El archivo de documentos es `./corpus/doc-text.trec` y está en un formato con tags para el número de doc (`<docno>`) y el texto a continuación.
[8] Utilice los datos de entrenamiento y de evaluación provistos en: https://github.com/tolosoft-academia/RI_2025/blob/main/data/languageIdentificationData.tar.gz
[9] https://pypi.org/project/langdetect/

Propiedades del Texto[10]

7) En este ejercicio se propone verificar la predicción de ley de Zipf. Para ello, descargue desde Project Gutenberg el texto del Quijote de Cervantes[11] 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 módulo NumPy[12]. 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?

8) Usando los datos del ejercicio anterior y de acuerdo a la ley de Zipf, calcule la cantidad de palabras que debería haber en el 10%, 20% y 30% del vocabulario. Verifique respecto de los valores reales. Utilice esta aproximación para podar el vocabulario en los mismos porcentajes e indique qué porcentaje de la poda coincide con palabras vacías[13]. Extraiga las palabras podadas que no son *stopwords* y verifique si, a su criterio, pueden ser importantes para la recuperación.

9) 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.

---

[10] Los siguientes ejercicios se deben realizar en una *notebook* IPython para trabajar en un entorno interactivo con capacidades gráficas.
[11] http://www.gutenberg.org/cache/epub/2000/pg2000.txt (en UTF-8)
[12] https://docs.scipy.org/doc/numpy-1.15.0/reference/generated/numpy.polyfit.html
[13] Use las palabras vacías provistas en la librería NLTK

Bibliografía

[1] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. Modern Information Retrieval: The Concepts and Technology Behind Search. Addison-Wesley Publishing, USA, 2nd edition, 2008. Cap. 7.

[2] Gregory Grefenstette and Pasi Tapanainen. What is a word, what is a sentence? problems of tokenization. In Rank Xerox Research Centre, pages 79–87, 1994.

[3] Le Quan Ha, Darryl Stewart, Philip Hanna, and F Smith. Zipf and type-token rules for the english, spanish, irish and latin languages. Web Journal of Formal, Computational and Cognitive Linguistics, 1 (8):1–12, 1 2006.

[4] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Sch ¨utze. Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA, 2008. Cap.6.

[5] Gabriel Tolosa, Fernando Bordignon. Introducción a la Recuperación de Información. Conceptos, modelos y algoritmos básicos. Laboratorio de Redes de Datos. UNLu, 2005. Cap. 3.