UNIVERSITAT POLITÈCNICA DE CATALUNYA

GRADO EN INGENIERÍA INFORMÁTICA

CERCA I ANÀLISI D'INFORMACIÓ MASSIVA

Q1 Course 2020-2021

Indexing and Similarity

GRUP 21

Autores:
Daniel CANO CARRASCOSA

0 INTRODUCCIÓN

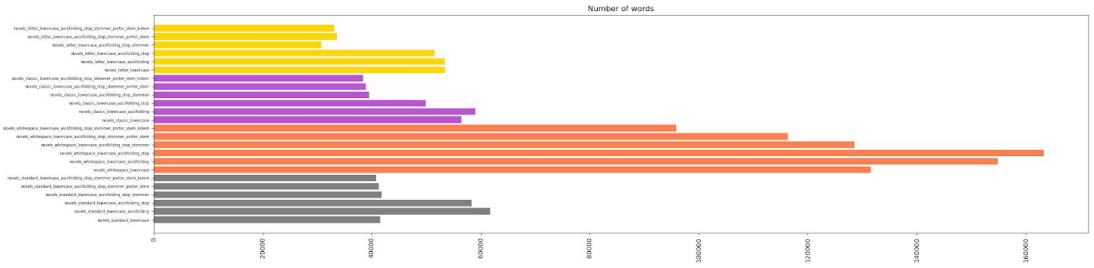
Para esta entrega el objetivo en mente es en primer lugar analizar las características de los documentos cuando cambiamos los posibles filtros y formas de tokenización en los documentos, e implementar y analizar la similitud por cosinos.

1 INDEXANDO

Para esta sección en primera instancia me he planteado la posibilidad de ir a mano, configuración a configuración probando y sacar conclusiones, pero dado que era una tarea mecánica, he decidido finalmente cambiar la implementación del código (más detalle en este mismo con comentarios) para tener la posibilidad de automatizar todas las posibilidades, y ya que estábamos en ello analizar tanto el número de palabras diferentes que obtenemos como la frecuencia máxima.

1.1 CANTIDAD DE TOKENS OBTENIDOS

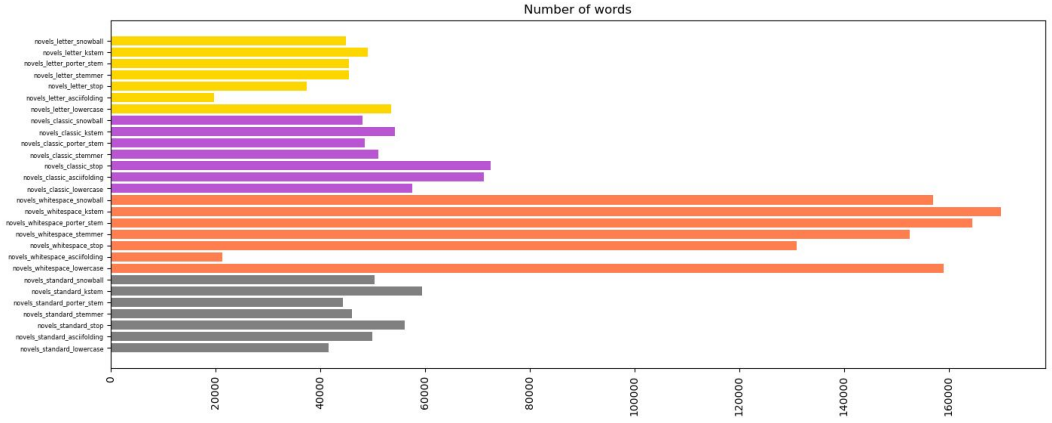
En primer lugar vamos a analizar el número de tokens o de palabras obtenidas con los diferentes métodos de tokenización combinados con diferentes filtros.



Para este caso tenemos que las secciones se distribuyen según el token usado los cuales son **letter**, **classic**, **whitespace**, **standard**. Y lo que podemos destacar de sobremanera es como cuando usamos el token whiteSpace, da igual que filtro añadamos o cuantos añadamos (quizás en la imagen no se ve pero hay diversos, como he comentado las imagenes estan

disponibles con la entrega) , que se puede ver como obtenemos significativamente más palabras que en todos los otros casos en los que usamos los otros filtros, lo cual es razonable ya que para el caso de letter eliminamos todo por cada símbolo que no es una palabra, por classic se basa en reglas gramaticales por lo que quizás haya conjuntos de palabras que con el whitespace se separarían y el standard únicamente se encarga principalmente de símbolos de puntuación, con lo que para todos dejando de lado el tokenizer whitespace el resultado es el esperado.

No obstante para confirmar esto he vuelto a ejecutar aplicando un único filtro con todos los filtros posibles para verificar que esto era así y por el resultado obtenido se puede interpretar que las conclusiones previas son ciertas, ya que es solo cuando dejamos todas las palabras únicamente alfabéticas que el whitespace se nivela con los demás tokenizers.



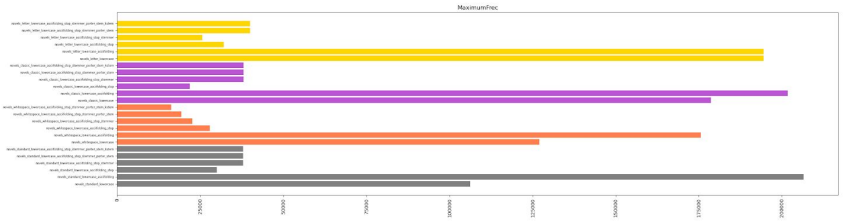
Nota: Conforme se envía el documento con el código, se complementará con estas imágenes para que puedan ser visualizadas correctamente, ya que debido a la cantidad de archivos y posibles combinatorias de opciones, he obtenido resultados como el mostrado (que no es visible desde el documento a no ser que ocupe 2 páginas) o bien resultados que ni siquiera minimizados cabrían de forma razonable en el documento, motivo del cual en el código se comenta todo en detalle cómo poder, si se quiere, modificar el código para un mayor análisis

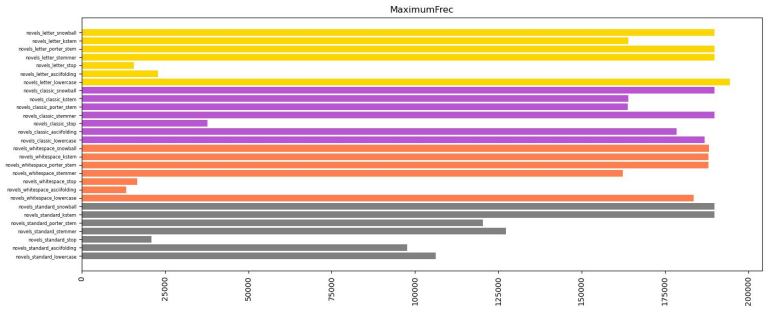
1.2 FRECUENCIA MÁXIMA

Finalmente y para analizar las frecuencias ya que tenía curiosidad sobre su comportamiento, me he dispuesto a analizarlas, y me he quedado bastante sorprendido ya que no importa que token o filtro apliquemos, dejando de lado algunos casos, la mayoría son bastante similares pese a sus diferencias mostradas previamente en cuanto al número de palabras, lo que da a entender dos cosas:

1. Las palabras más frecuentes son iguales en prácticamente todos (pese a que como la palabra es “the” al aplicar el filtro stopwords cae repentinamente)
2. Da igual el número total de palabras que haya en nuestro texto, dejando de lado los filtros asciifolding y stop (lo que da a entender de dónde vienen la mayoría de caracteres, ya que la caída es bastante pronunciada), en la mayoría hay, incluso con las de whitespace que recordemos que tienen muchisimas mas palabras, una riqueza progresiva de palabras, es decir no importa que whitespace tenga más palabras que otros textos, ya que al crecer la cantidad de estas también está creciendo la riqueza léxica.

A continuación se muestran las dos imágenes de las máximas frecuencias, la primera combinando varios filtros simultáneos, donde sí que se puede ver una caída respecto más filtros se aplican, pero como he dicho previamente todos caen de forma similar (misma riqueza de palabras, pese a gran diferencia en cantidad), y en el segundo solo aplicando un filtro vemos cómo se mantiene prácticamente igual para todas.





2 SIMILITUD

Para esta sección comentaré la implementación del código para la relación entre dos documentos dado el vector tf-idf, así como problemas que haya encontrado, que hace el código y extensiones por mi parte

2.1 ANALISIS DEL CODIGO

En cuanto a esta sección analizaré brevemente el código, aunque no entraré mucho en detalle ya que todo está comentado en el mismo. El código trata esencialmente de dado un path a una carpeta y dos ficheros del mismo calcular la verosimilitud, y para esto el programa inicial genera un vector de las frecuencias de todas las palabras en cada documento, así como por cada palabra de cada uno de los dos el número de textos en los que aparece, lo cual se usará para calcular el tf-idf y así poder aplicar la similitud por coseno.

2.2 PROBLEMAS ENCONTRADOS Y EXTENSIONES

En cuanto a la parte de implementar el código faltante, la verdad no he tenido ningún problema ya que prácticamente todo estaba hecho, y con la ayuda de unas pocas funciones de numpy y unas listas de compresión completar el código no ha sido realmente un esfuerzo, por lo que dada la situación he decidido ampliarlo, implementando las siguientes características manteniendo todas las anteriores:

1. Parámetro opcional --path_all_files, si se indica el programa analiza la similaridad todas las posibles combinatorias de dos archivos, esto lo he implementado para en vez de ir archivo a archivo comparando a ver si encontraba algo interesante, dejar que el mismo programa lo hiciera automáticamente y ver si después de un periodo de tiempo de espera obtenía buenos resultados, pero esta combinatoria de posibilidades viene dada por la fórmula $\frac{n!}{(n-2)!2}$ por lo que dado el hecho de que la combinatoria era enorme (para arxiv hay 200.000.000 de posibilidades) he decidido añadir dos parámetros extra.

2. Si se indica el parámetro --path_all_files entonces es obligatoria indicar o bien --prog o --iter, lo cual hace printear por pantalla el porcentaje total del análisis de todos las posibilidades, lo cual se puede saber por la fórmula anterior.
3. Y por ultimo el parámetro --stop permite que cada --iter iteraciones, o cuando se haya cierto un progreso de --prog porcentaje, se da la oportunidad de parar el programa conservando todos los resultados en ficheros indicando los nombres y la similitud.

Gracias a esto he sido capaz de detectar casos como en news donde la similaridad entre los ficheros 0000000 y 0000900 es del 99% nada mas y nada menos!