Explotación de la Información
Práctica 3 - Buscador

Alejandro Jesús Aliaga Hyder
ajah1@alu.ua.es
48765284V

Índice

Cambios en la práctica

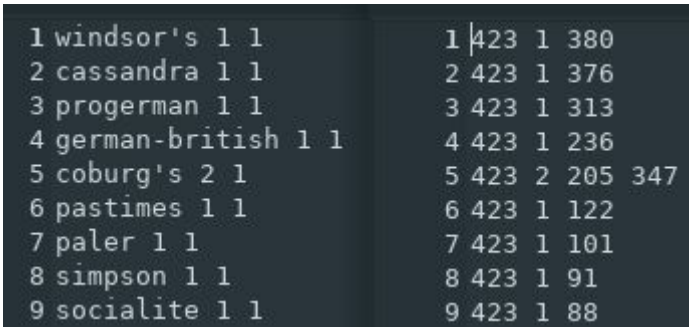
vector de posiciones

En la clase InfTermDoc se usa una lista para almacenar las posiciones en las que se encuentra el término, a la hora de leer la lista de la indexación guardada en ficheros el programa no paraba de aumentar la memoria. Por ello a la lista posterm le he cambiado la estructura por un vector, dado que la lista no guarda la memoria de forma consecutiva moverse demasiado por la memoria a la hora de guardar las posiciones.

¿Porque se ha usado un vector y no otra estructura de datos?
El vector reserva memoria de forma geométrica, a la hora de indexar el corpus puede ser un problema que se reserve el doble de memoria de la necesaria, para evitar ese problema hago uso de un vector auxiliar ya que al copiar estos datos en el objeto InfTermDoc la capacity del vector se ajusta a la del size, por tanto la memoria que se reserva de más en el vector auxiliar se acaba liberando según se itera en al indexación. En el caso de leer la indexación guardada en disco se evita el problema ya que en los ficheros se almacena el número de veces que aparece el término en el corpus lo que permite hacer reserve del vector.

Forma de guardar / leer la indexación

La forma que usaba tanto de leer como guardar la indexación ya no me sirven en este práctica dado que usaba mucha memoria e iteraba demasiado sobre los documentos ya que usaba la sobrecarga del operador salida en los ficheros. Para ello he modificado las sobrecargas de los operadores salida de las clases haciendo que los datos solo se separen por un espacio lo que permite leer parte de ellos de “golpe”. Además el fichero el índice es demasiado grande como para almacenar todo en un fichero por ello la lista de posiciones se almacena en otro fichero. Un ejemplo de cómo queda el índice en memoria es el siguiente:



Memoria usada

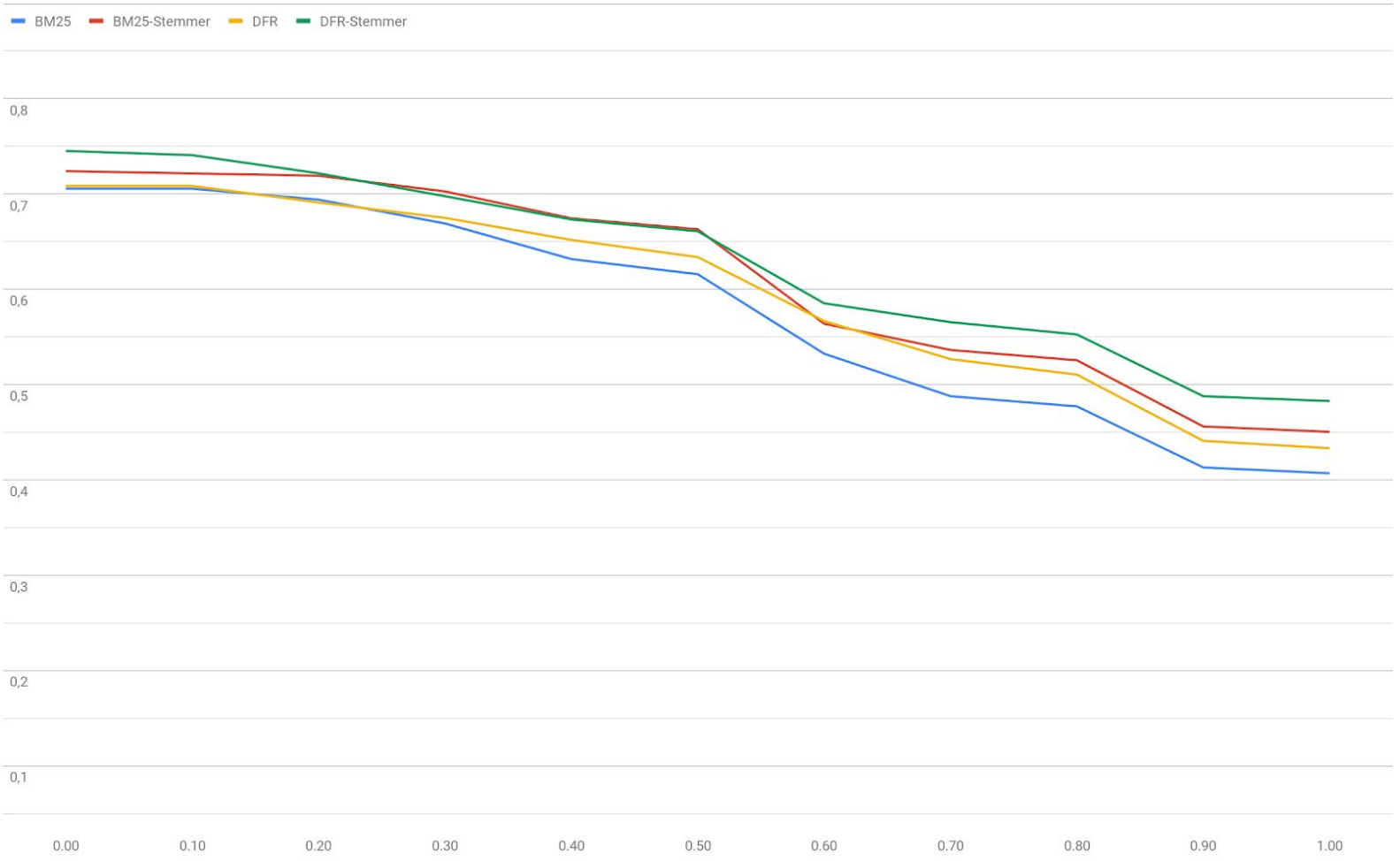
He ejecutado el valgrind en el main que he usado para obtener la salida del trec_eval y como resultado no dejo memoria sin liberar.

```
==4553==
==4553== HEAP SUMMARY:
==4553==     in use at exit: 0 bytes in 0 blocks
==4553==   total heap usage: 2,039,303 allocs, 2,039,303 frees, 133,094,150 bytes allocated
==4553==
==4553== All heap blocks were freed -- no leaks are possible
==4553==
==4553== For counts of detected and suppressed errors, rerun with: -v
==4553== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Tabla de los datos obtenidos por trec_eval

	Precision Averages						
	BM25		**BM25-STEMMER**		**DFR**		**DFR-STEMMER**
0.00	0,7057		0,7241		0,7086		0,7453
0.10	0,7057		0,7217		0,7086		0,7409
0.20	0,6942		0,7193		0,6912		0,7218
0.30	0,6692		0,7028		0,6751		0,6978
0.40	0,6318		0,6746		0,6519		0,6734
0.50	0,6159		0,6631		0,6339		0,661
0.60	0,5325		0,564		0,5668		0,5854
0.70	0,4879		0,5364		0,5268		0,5656
0.80	0,4772		0,5256		0,5105		0,5527
0.90	0,4131		0,4561		0,441		0,4879
1.00	0,407		0,4505		0,4333		0,4828

Representación gráfica de los resultados



Tras comparar los resultados para cada fórmula de similitud los resultados obtenidos muestran que el método por el que se obtiene mejor acierto es el DFR siendo bastante la diferencia entre ambos modelos. En cuanto al preprocesado del texto y las preguntas con stemming, los resultados obtenidos son mejores respecto a la versión sin aplicarlo. Aunque al principio DFR y BM25 con stemming se asemejan mucho al final el DFR acaba obteniendo mejores resultado siendo bastante notoria la diferencia con el BM25 sin stemming.