Prácticas Recuperación de Información.
Grado en Ingeniería Informática.

P2. Indexación, búsqueda y evaluación sobre la colección NPL

En primer lugar debe indexarse la colección NPL.
http://ir.dcs.gla.ac.uk/resources/test_collections/npl/

En esta práctica es necesario en primer lugar parsear el archivo (doc.txt) que contienen los documentos NPL y también los contenidos para cada documento. Deben indexarse dos campos: **DocIDNPL** para indexar el número de documento NPL y **Contents** con el contenido. Ambos campos ponedlos como indexados y almacenados. Además de las opciones de indexación básicas (indexación no concurrente y se puede suponer un sólo archivo que contiene los documentos NPL) de P1, será necesario indicar el modelo de recuperación de información (usaremos los modelos de lenguaje con suavización de Jelinek-Mercer y Dirichlet, clases de Lucene LMJelinekMercerSimilarity y LMDirichletSimilarity). Esto es necesario porque no usaremos la Similarity class actual por defecto de Lucene que es BM25.

Indexación.
La clase principal se llamará **IndexNPL** y tendrá como argumentos:
-openmode openmode (el open mode será append, create, o create_or_append)
-index pathname (ruta de la carpeta que contiene o contendrá el índice)
-docs pathaname (ruta de la carpeta que contiene los docs de la colección)
-indexingmodel cuyos valores posibles son jm lambda | dir mu
-analyzer analyzer (analyzer es uno de los built-in analyzers de Lucene)

Recordar que con el método setSimilarity del IndexWriterConfig hay que indicar la Similarity class al crear el índice.

Búsqueda y evaluación de queries.
A continuación se trata de construir un buscador y evaluador para las queries de la colección con las opciones que se detallan usando el archivo de queries (query-txt) y los juicios de relevancia (rlv-ass). Las queries se procesarán con query parser sobre el campo Contents. Las métricas a calcular serán P@n, Recall@n, RR y AP@n. P@n se computa dividiendo por n, el número de relevantes recuperados en las primeras n posiciones del ranking; Recall@n se computa dividiendo por el total de relevantes para la query, el número de relevantes recuperados en las primeras n posiciones del ranking. El AP@n (Average Precision hasta el corte n) se computa calculando la precisión cada vez que aparece un documento relevante en el ranking hasta llegar al corte n en el ranking, sumando esas precisiones y dividiendo por el número de relevantes por query. RR (Reciprocal Rank) es el inverso de la posición del ranking donde se encuntra el primer relevante y MRR (Mean Reciprocal Rank) es el promedio. Las métricas se computan para cada query y se promedian para las queries evaluadas. El promedio de AP@n para un número de queries se llama MAP@n (Mean Average Precision). Las queries que no tienen relevantes se ignoran en la evaluación. Los promedios para un número de queries de P@n y Recall@n no tienen nombre especial. Para cada query se visualizará la query, el top m de documentos, y para cada documento se visualizarán todos los campos del índice, el score del documento y una marca que diga si es relevante según los juicios de relevancia, y las métricas individuales para cada query. Finalmente se mostrarán las métricas promediadas.

La clase **SearchEvalNPL** se ocupará de la búsqueda y evaluación con estas opciones:

-search jm lambda | dir mu (indica el modelo de RI para la búsqueda y el valor del parámetro de suavización)
-indexin pathname (ruta de la carpeta que contiene el índice).
-cut n (n indica el corte en el ranking para el cómputo de las metricas P, R, RR, AP). Si no hay relevantes en el corte n para una query, P, R y AP por su definición valen cero. RR no estaría definido para este caso pero lo justo para hacer promedios con otras queries es considerala también cero).
-top m (visualiza el top m de documentos del ranking)
-queries all | int1 | int1-int2 (lanza y evalua la query int1, las queries en el rango int1-int2, ambas inclusive, o todas la queries, intx son los enteros que indentifican las queries en el archivo de queries)

Recordar que hay que indicarle al IndexSearcher con el método setSimilarity la Similarity Class y el valor del parámetro de suavización. Recordad también se debe usar siempre el mismo analizador en indexación y procesado de las consultas para aplicar en ambos casos los mismos procesos de tokenización, stop words, stemming, etc. Es importante observar el resultado del Query Parser antes de lanzar la query. A modo de ejemplo, si el texto de la query tiene AND o NOT y el Query Parser lo trata antes de que el analizador lo pase a minúsculas, en el caso de que ese analizador pase a minúsuculas, daría lugar a una query con requerimientos booleanos que no se pretendían. Eso sería de fácil solución usando el método de la clase String que pasa el texto a minúsculas, pero es necesario observar el resultado por si hay otras incidencias a resolver. También observad si hay algún salto de línea u otros caracteres que puedan confundir el parseado. Se acepta hacer cualquier modificación manual del archivo de queries para resolver este tipo de incidencias.

El top m de resultados visualizados por pantalla se volcarán también a un archivo con nombre, a modo de ejemplos: npl.jm.10.hits.lambda.0.2.q1-20.txt (los primeros 10 resultados del modelo JM con lambda=0.2 y para las queries 1 al 20), npl.jm.10.hits.lambda.0.0.qall.txt (los primeros 10 resultados del modelo JM con lambda=0.0, es decir sin suavización, y para todas las queries), npl.dir.10.hits.mu.1500.qall.txt (los primeros 10 resultados del modelo LM Dirichlet para mu=1500 para todas las queries)
Con los resultados se producirá también un archivo .csv con una fila por query, la primera columna para identificar la query, una columna por métrica, la primera fila para indentificar las métricas indicando también el corte n para la que se computó, y una última fila para los promedios. El nombre del archivo .csv seguirá el patrón, a modo de ejemplo, npl.jm.10.cut.lambda.0.2.q1-20.csv (resultados para jm com lambda=0.2 computados para las métricas con corte 10 sobre las queries 1al 20), npl.dir.10.cut.mu.1500.q1-20.csv (resultados para Dirichlet mu=1500 computados para las métricas con corte 10 sobre las queries 1al 20).

Training & test.
La clase **TrainingTestNPL,** con la opción -evaljm se ocupará de encontrar el valor óptimo del párametro de suavización del modelo basado en Language Models JM, para un conjunto de queries de entrenamiento y de aplicar ese valores óptimo (modelo óptimo) a un conjunto de queries de test.

-evaljm int1-int2 int3-int4
-evaldir int1-int2 int3-int4 **(las opciones -evaljm -evaldir son mutuamente exclusivas)**
-cut n (n indica el corte en el ranking para el cómputo de la metrica)
-metrica P | R | MRR | MAP (indica la métrica computada y optimizada en el corte n)
-indexin pathname (ruta de la carpeta que contiene el índice)

Sobre el índice indicado en pathname se lanzan las queries con el modelo LM Jelinek-Mercer (Dirichlet) de int1 a int2 y se evalúa la métrica en el corte n para esas queries. Se repite el proceso

moviendo lambda con los valores 0.0, 0.1 0.2, … , 1.0, (mu 0, 200, 400, 600, 800, 1000, 1500, 2000, 2500, 3000, 4000). Con el mejor valor de lambda (mu) en training se lanzan las queries de test de int3 a int4 y se informa del valor de la métrica con corte n obtenido para las queries de test y su promedio. Debe construirse un archivo .csv con los resultados del proceso de training con una fila por query, una columna por valor de lambda (mu), una fila cabecera informando de los valores de lambda (mu), la primera fila primera columna para informar de la métrica y el corte, y una última fila con los promedios. El nombre de este fichero será a modo de ejemplo npl.jm.training.1-20.test.21-30.map10.training.csv (muestra los resultados de training sobre las queries 1 al 20 con los valores de lambda JM optimizados para MAP@10). Debe construirse otro archivo con los resultados del proceso de test: una fila por query, una columna para indicar el valor de la métrica, una fila cabecera informando del nombre de la métrica, la primera fila primera columna para indicar el valor del lambda en test (lamda óptimo en training), y una última fila con los promedios. El nombre de este archivo a modo de ejemplo será npl.jm.training.1-20.test.21-30.map10.test.csv (muestra los resultados de test sobre las queries 21-30 para el modelo JM cuyo lambda se optimizó en las queries 1 al 20, tanto la optimización y el test con MAP@10).

-evaljm y -evaldir, después de producir los archivos csv también volcarán su contenido por pantalla.

Significancia estadística

La clase **Compare** se ocupa de realizar un test de significancia estadística sobre los resultados de dos modelo sobre las mismas queries de test. Observe que si dos modelos tienen parámetros, para una comparación justa debieran ser optimizados sobre las mismas queries de training. Pero en una experimentación pudiera interesar estudiar el efecto de la cantidad o naturaleza de datos de training y comparar resultados de modelos entrenados sobre distintos datos de training.

-test t|wilcoxon alpha (test de significancia estadística -t-test o Wilcoxon- y nivel de significancia alpha)
-results results1.csv results2.csv (results1 y results2 son archivos de resultados obtenidos con **TrainingTestNPL** para la misma métrica y sobre las mismas queries de test). A modo de ejemplo:
-results npl.jm.training.1-20.test.21-30.map10.test.csv npl.dir.training.1-20.test.21-30.map10.test.csv
Por tanto debe chequearse la coincidencia de los nombres de los archivos en la parte que interesa y se da por supuesto que los archivos .csv están bien construidos.

Se debe lanzar un test de significancia estadística (t-test o Wilcoxon) con nivel de significancia alpha e informar del resultado del test y el p-valor. Podéis usar la clase TTest, método pairedTTest o método tTest one sample; clase WilcoxonSignedRankedTest, método wilcoxonSignedRankedTest, de la librería matemática de Apache Commons u otra librería que uséis correctamente.

Dense Retrieval

La clase **DenseRetrieval** se ocupará de implementar un modelo de Dense Retrieval utilizando los clases y métodos usados para ello en las demos de Lucene IndexFiles y SearchFiles. Una vez obtenido ese modelo debe compararse con el mejor modelo obtenido en las secciones anteriores. **Esta funcionalidad es opcional**: puede obtenerse la máxima puntuación de P2 sin esta funcionalidad. Para los que se decidan a implementarla es conveniente que discutan con los profesores de prácticas su implementación antes de hacerla.

Entrega P2

Las prácticas se entregarán de manera similar a P1. Tenéis que crear un proyecto Maven **`mri-searcher`** y entregaréis por GitHub Classroom el archivo **pom.xml** y la carpeta **src** sin incluir archivos no necesarios para la entrega en la carpeta src.

En el proyecto deben aparecer cuatro clases con método main() que se corresponden con las partes de la práctica y con nombres **IndexNPL , SearchEvalNPL, TrainingTestNPL, Compare y DenseRetrieval** de forma que se puedan construir un *runnable jar* con cada parte.
-En las defensas parciales y final, el comportamiento de la práctica tiene que ser correcto y debe ser eficiente y se pedirán cambios que deberán implementarse en el aula y horario de prácticas.
-SE APLICAN LAS MISMAS NORMAS ESTABLECIDAS PARA P1